APPENDIX

STOCK MARKET PREDICTION WITH LSTM MODEL

1. INSTALATION AND IMPRORTING LIBRARIES

In [ ]:
```
#pip install tensorflow
```

In [ ]:
```
#pip install keras

```

In [ ]:
```
#pip install yahoo_fin

```

In [2]:
```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from math import sqrt
import tensorflow as tf
import matplotlib as mpl
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller
from yahoo_fin.stock_info import *
import seaborn as sns

%matplotlib inline
```

```
WARNING:tensorflow:From D:\WORK\software\Annaconda\Lib\site-packages\keras\src
\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated.
Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2. 'IBM' STOCK DATA FROM YAHOO FINANCIAL API

In [3]:
```
ibm_data = monthly_data = get_data("IBM",start_date = '01/01/1995', interval

ibm_data = ibm_data.reset_index()


ibm_data = ibm_data.rename(columns={"index": "DATE"})

ibm_data['DATE'] = pd.to_datetime(ibm_data['DATE'])
ibm_data['DATE'] = ibm_data['DATE'].dt.strftime('%Y-%m')
ibm_data = ibm_data.drop(['ticker'], axis=1)
ibm_data = ibm_data[['DATE', 'close']]

ibm_data = ibm_data.drop_duplicates()
ibm_data = ibm_data[~ibm_data.index.duplicated(keep='first')]


ibm_data.set_index('DATE', inplace=True)
ibm_data.index = pd.to_datetime(ibm_data.index)
ibm_data.head()
```

Out[3]:

DATE	close
1995-01-01	17.238289
1995-02-01	17.985182
1995-03-01	19.628345
1995-04-01	22.615917
1995-05-01	22.227533

In [4]:
```

missing_values = ibm_data['close'].isna().sum()
print(f"Number of missing values in 'close': {missing_values}")

```

```
Number of missing values in 'close': 0
```

In [5]:
```
ibm_data.head()
```

Out[5]:

DATE	close
1995-01-01	17.238289
1995-02-01	17.985182
1995-03-01	19.628345
1995-04-01	22.615917
1995-05-01	22.227533

In [6]:
```
def test_stationarity(timeseries):
    # Dickey-Fuller test
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)' % key] = value
    print(dfoutput)

# Perform stationarity test on the 'close' column of the dataset
test_stationarity(ibm_data['close'])

```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.017504
p-value                          0.278951
#Lags Used                       1.000000
Number of Observations Used    347.000000
Critical Value (1%)             -3.449337
Critical Value (5%)             -2.869906
Critical Value (10%)            -2.571227
dtype: float64
```

In [7]:
```
# Applying first differencing to the 'close' column to make the data station
ibm_data['close_diff'] = ibm_data['close'].diff()

# Remove any NaN values created by differencing
data_diff = ibm_data.dropna()
# Re-run the Dickey-Fuller test on the differenced data
test_stationarity(data_diff['close_diff'])

```

```
Results of Dickey-Fuller Test:
Test Statistic                 -20.363115
p-value                          0.000000
#Lags Used                       0.000000
Number of Observations Used    347.000000
Critical Value (1%)             -3.449337
Critical Value (5%)             -2.869906
Critical Value (10%)            -2.571227
dtype: float64
```

In [8]:
```
mean_diff = ibm_data['close_diff'].mean()

# Fill the NaN values in 'close_diff' with this mean value
ibm_data['close_diff'].fillna(mean_diff, inplace=True)




ibm_data

```

Out[8]:

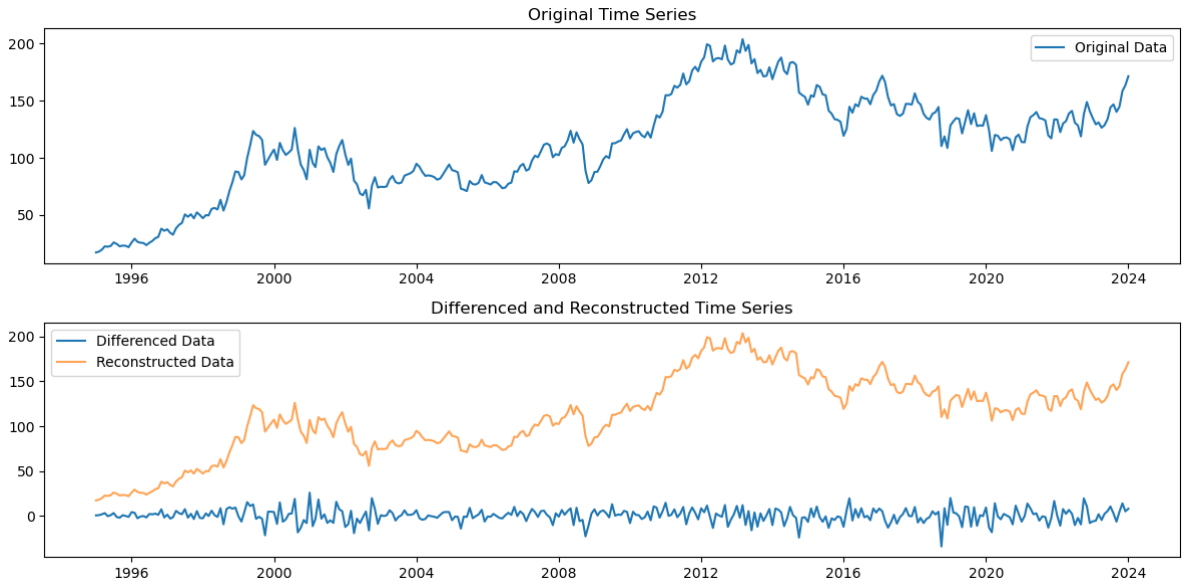
DATE	close	close_diff
1995-01-01	17.238289	0.443223
1995-02-01	17.985182	0.746893
1995-03-01	19.628345	1.643164
1995-04-01	22.615917	2.987572
1995-05-01	22.227533	-0.388384
...	...	...
2023-09-01	140.300003	-6.529999
2023-10-01	144.639999	4.339996
2023-11-01	158.559998	13.919998
2023-12-01	163.550003	4.990005
2024-01-01	171.479996	7.929993

349 rows × 2 columns

In [ ]:
```


```

In [9]:

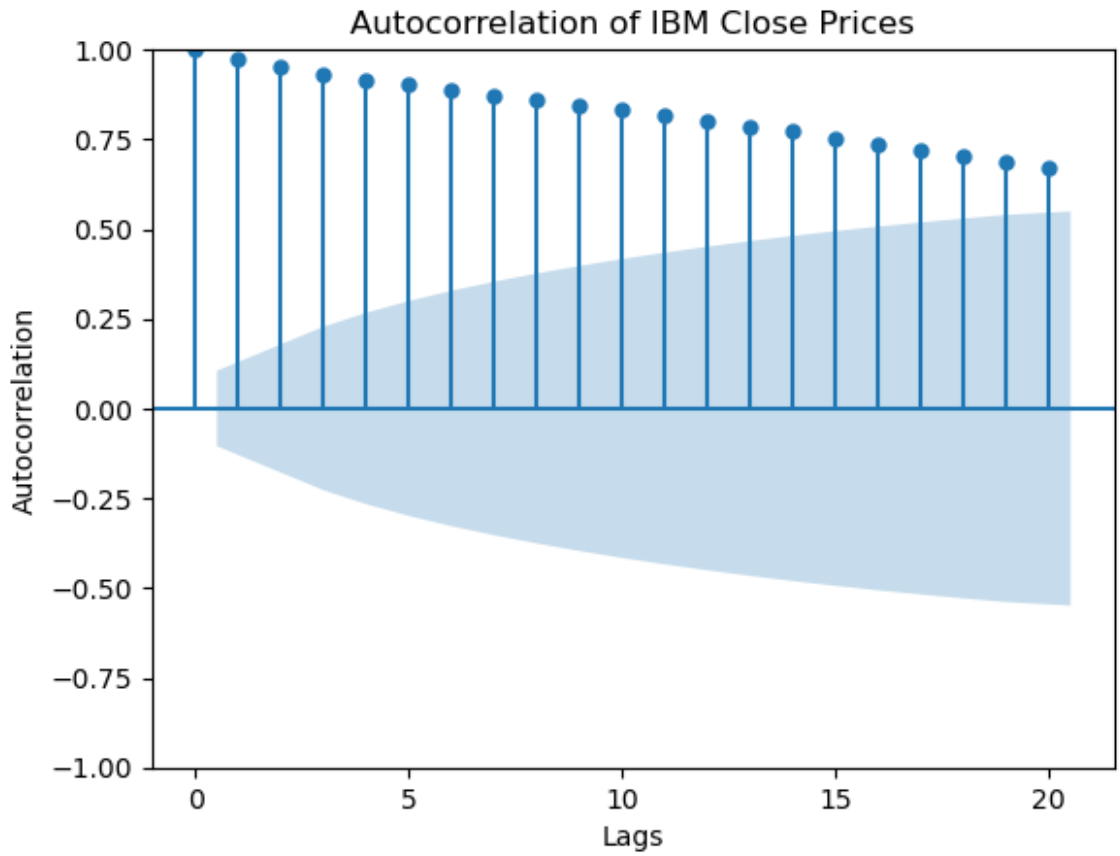
```
original_data =ibm_data['close']
differenced_data =ibm_data['close_diff']
reconstructed_data =ibm_data['close']
# Plotting
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(original_data, label='Original Data')
plt.title('Original Time Series')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(differenced_data, label='Differenced Data')
plt.plot(reconstructed_data, label='Reconstructed Data', alpha=0.7)
plt.title('Differenced and Reconstructed Time Series')
plt.legend()

plt.tight_layout()
plt.show()
```



In [10]:

```
# Plotting the Autocorrelation for 'close_diff'
plt.figure(figsize=(12, 6))
plot_acf(ibm_data['close'], lags=20)
plt.title('Autocorrelation of IBM Close Prices' )
plt.xlabel('Lags')
plt.ylabel('Autocorrelation')
plt.show()
```

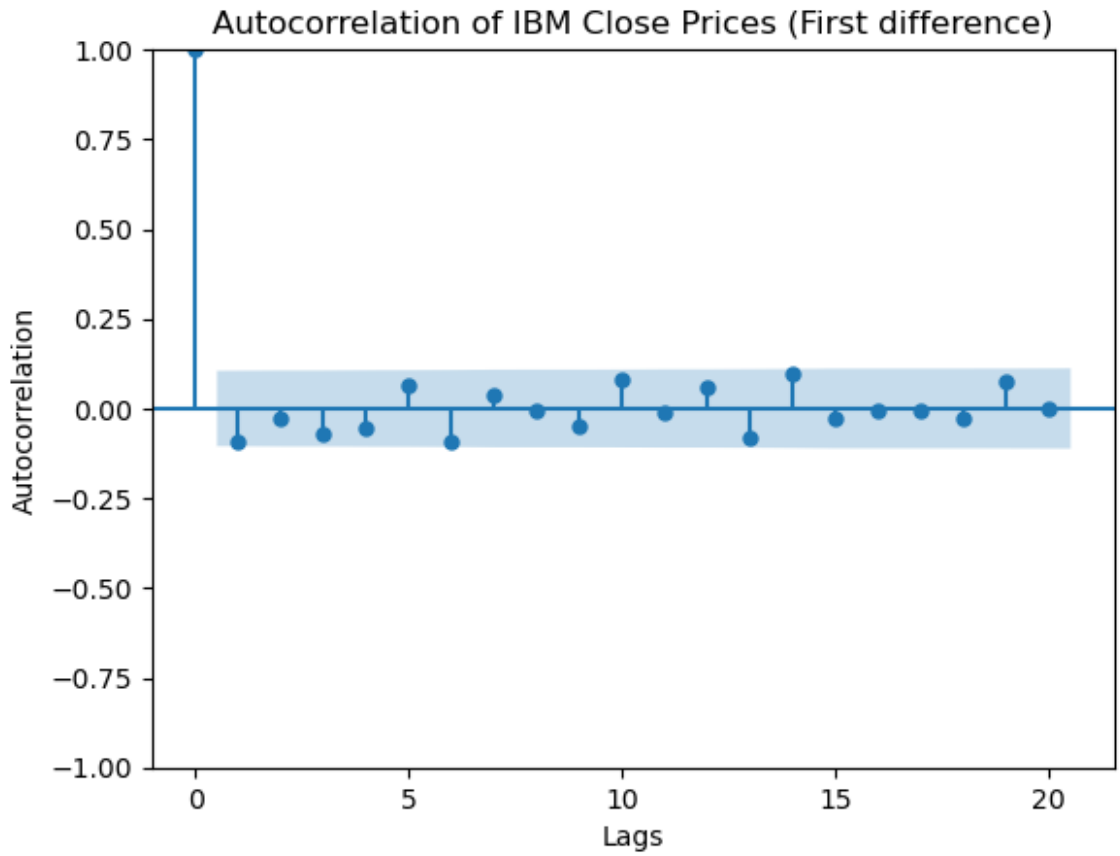
```
<Figure size 1200x600 with 0 Axes>
```



In [11]:

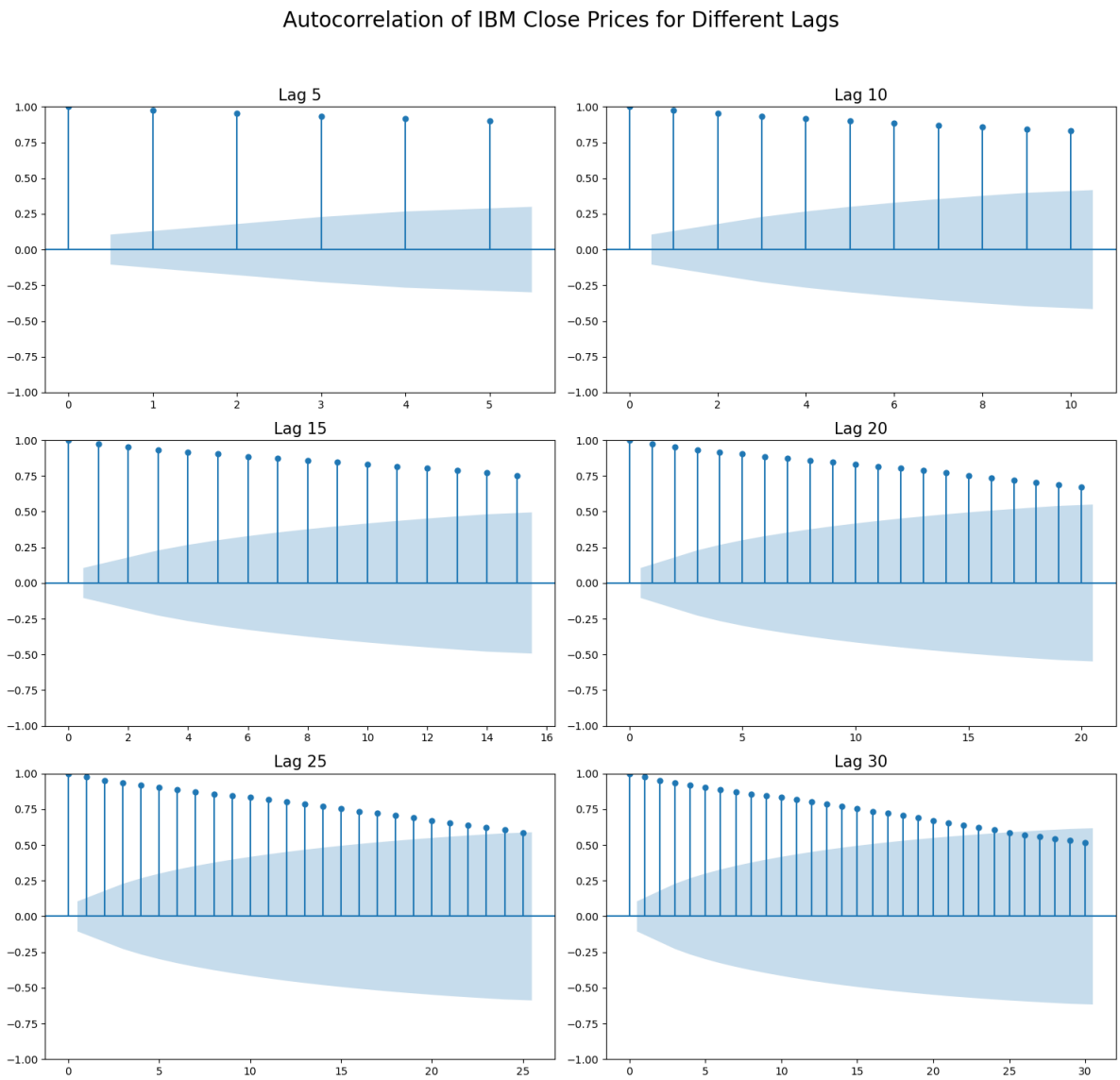
```
# Plotting the Autocorrelation for 'close_diff'
plt.figure(figsize=(12, 6))
plot_acf(ibm_data['close_diff'], lags=20)
plt.title('Autocorrelation of IBM Close Prices (First difference)' )
plt.xlabel('Lags')
plt.ylabel('Autocorrelation')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



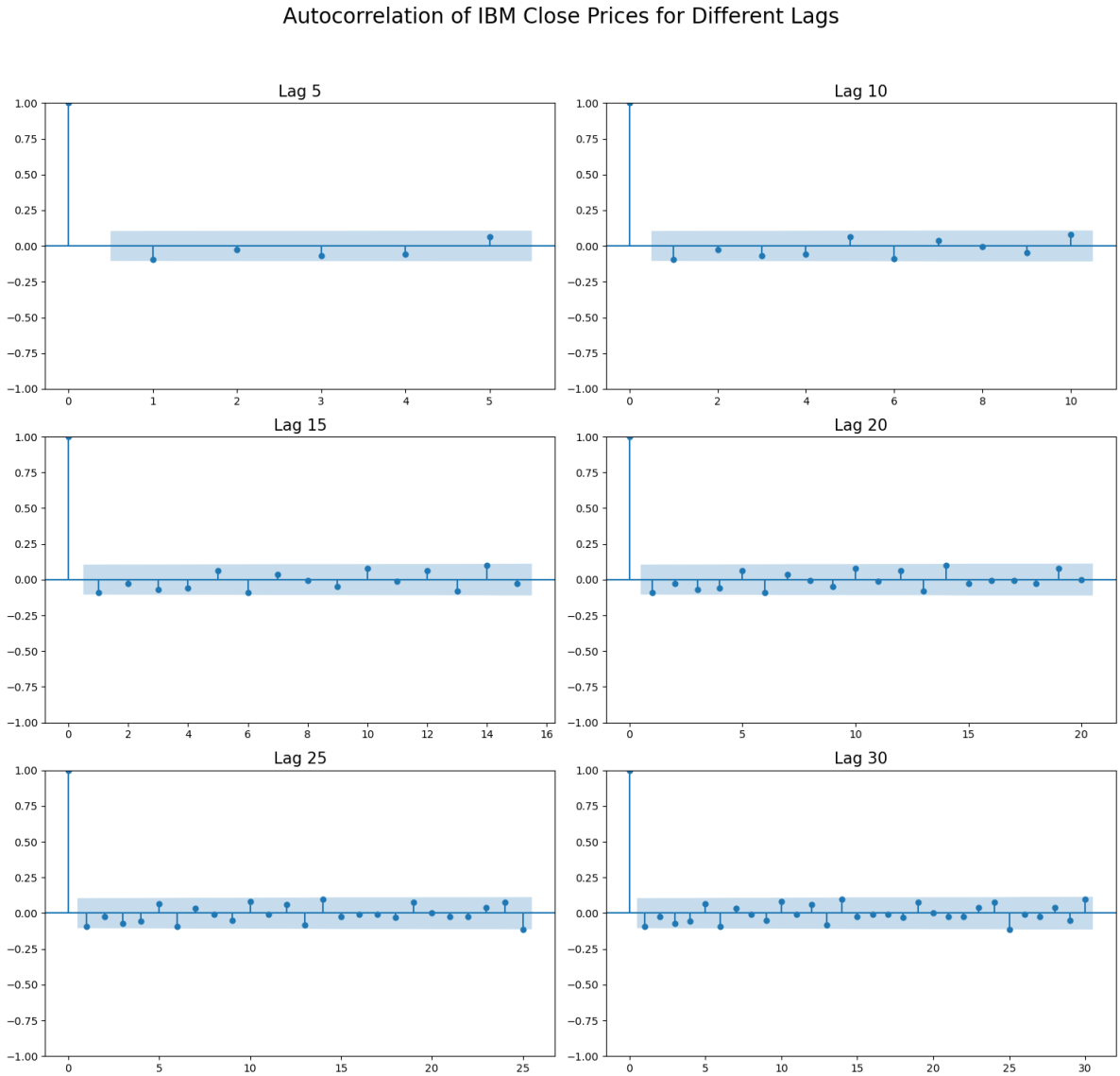
In [12]:

```
# Define the number of lags you want to plot
lags_to_plot = [5, 10, 15, 20, 25, 30]

# Determine the number of rows and columns based on the length of lags_to_pl
num_plots = len(lags_to_plot)
num_rows = num_plots // 2 + num_plots % 2
num_cols = 2

# Create a grid of subplots with a specified figure size
fig, ax = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, 15))
fig.suptitle('Autocorrelation of IBM Close Prices for Different Lags', fonts

# Loop over the lags_to_plot list and create autocorrelation plots
for i, lag in enumerate(lags_to_plot):
    row, col = i // num_cols, i % num_cols
    plot_acf(ibm_data['close'], lags=lag, ax=ax[row, col], alpha=0.05)
    ax[row, col].set_title(f'Lag {lag}', fontsize=15)

# Adjust layout to prevent overlapping plots
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

# Show the plot
plt.show()

```



In [13]:

```
# Define the number of lags you want to plot
lags_to_plot = [5, 10, 15, 20, 25, 30]  # Ensure this list has the same numb

# Determine the number of rows and columns based on the length of lags_to_pl
num_plots = len(lags_to_plot)
num_rows = num_plots // 2 + num_plots % 2
num_cols = 2

# Create a grid of subplots with a specified figure size
fig, ax = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, 15))
fig.suptitle('Autocorrelation of IBM Close Prices for Different Lags', fonts

# Loop over the lags_to_plot list and create autocorrelation plots
for i, lag in enumerate(lags_to_plot):
    row, col = i // num_cols, i % num_cols
    plot_acf(ibm_data['close_diff'], lags=lag, ax=ax[row, col], alpha=0.05)
    ax[row, col].set_title(f'Lag {lag}', fontsize=15)

# Adjust layout to prevent overlapping plots
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

# Show the plot
plt.show()

```



5. SMA PREDICTION

In [16]:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt


# Initialize the scaler
scaler = MinMaxScaler(feature_range=(0, 1))

# Scale the close prices
ibm_data_scaled = scaler.fit_transform(ibm_data['close'].values.reshape(-1,

# Inverse scale the data to get the original prices back
ibm_data_unscaled = scaler.inverse_transform(ibm_data_scaled)

# Convert the unscaled data back to a DataFrame for convenience
ibm_data_unscaled_df = pd.DataFrame(ibm_data_unscaled, index=ibm_data.index,

# Define the window size for SMA and the proportion for training data
window_size = 5  # Adjust this window size as needed
train_proportion = 0.8

# Calculate the index to split the data
split_index = int(len(ibm_data_scaled) * train_proportion)

# Split the data into training and testing sets
train_data = ibm_data_unscaled_df.iloc[:split_index]
test_data = ibm_data_unscaled_df.iloc[split_index:]

# Calculate SMA for the train data
train_data['sma'] = train_data['close'].rolling(window=window_size).mean()

# Apply SMA to the test data
test_data['sma'] = test_data['close'].rolling(window=window_size).mean()

# Plotting the SMA predictions against the original close prices
plt.figure(figsize=(18, 9))
plt.plot(train_data.index, train_data['close'], color='blue', label='Train C
plt.plot(train_data.index, train_data['sma'], color='orange', label='Train S
plt.plot(test_data.index, test_data['close'], color='green', label='Test Clo
plt.plot(test_data.index, test_data['sma'], color='red', label='Test SMA')
plt.title('Train and Test: Close Price vs SMA')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.ylim(0, 200)  # Set y-axis limits from 0 to 200
plt.legend()
plt.show()

# Align the 'close' column to the length of the SMA-aligned data
test_data_aligned = test_data.dropna(subset=['sma'])

# Ensure both arrays have the same length now
assert len(test_data_aligned['sma']) == len(test_data_aligned['close']), "Le

# Now you can calculate the MSE with the aligned data
mse_error_sma = mean_squared_error(test_data_aligned['close'], test_data_ali
print('MSE error for SMA on Test Data: {:.5f}'.format(mse_error_sma))

```

```
C:\Users\DeLL\AppData\Local\Temp\ipykernel_1424\3265027757.py:33: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  train_data['sma'] = train_data['close'].rolling(window=window_size).mean()
C:\Users\DeLL\AppData\Local\Temp\ipykernel_1424\3265027757.py:36: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test_data['sma'] = test_data['close'].rolling(window=window_size).mean()
```


```
MSE error for SMA on Test Data: 69.84106
```

In [203]:
```
1 len(test_scaled)
```

Out[203]:
```
128
```

6. EMA MODEL PREDICTION

In [23]:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Initialize the scaler
scaler = MinMaxScaler(feature_range=(0, 1))

# Scale the close prices
ibm_data_scaled = scaler.fit_transform(ibm_data['close'].values.reshape(-1,

# Inverse scale the data to get the original prices back
ibm_data_unscaled = scaler.inverse_transform(ibm_data_scaled)

# Convert the unscaled data back to a DataFrame for convenience
ibm_data_unscaled_df = pd.DataFrame(ibm_data_unscaled, index=ibm_data.index,

# Define the proportion for training data
train_proportion = 0.8

# Calculate the index to split the data
split_index = int(len(ibm_data_scaled) * train_proportion)

# Split the data into training and testing sets
train_data = ibm_data_unscaled_df.iloc[:split_index]
test_data = ibm_data_unscaled_df.iloc[split_index:]

# Define the span for EMA calculation
span = 5  # This is similar to the 'window' parameter in SMA

# Calculate EMA for the train data
train_data['ema'] = train_data['close'].ewm(span=span, adjust=False).mean()

# Apply EMA to the test data
test_data['ema'] = test_data['close'].ewm(span=span, adjust=False).mean()

# Plotting the EMA predictions against the original close prices
plt.figure(figsize=(18, 9))
plt.plot(train_data.index, train_data['close'], color='blue', label='Train C
plt.plot(train_data.index, train_data['ema'], color='orange', label='Train E
plt.plot(test_data.index, test_data['close'], color='green', label='Test Clo
plt.plot(test_data.index, test_data['ema'], color='red', label='Test EMA')
plt.title('Train and Test: Close Price vs EMA')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.ylim(0, 200)  # Set y-axis limits from 0 to 200
plt.legend()
plt.show()

# Align the 'close' column to the length of the EMA-aligned data
test_data_aligned = test_data.dropna(subset=['ema'])

# Ensure both arrays have the same length now
assert len(test_data_aligned['ema']) == len(test_data_aligned['close']), "Le

# Now you can calculate the MSE with the aligned data
mse_error_ema = mean_squared_error(test_data_aligned['close'], test_data_ali
print('MSE error for EMA on Test Data: {:.5f}'.format(mse_error_ema))

```

```
C:\Users\DeLL\AppData\Local\Temp\ipykernel_1424\1934862544.py:35: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  train_data['ema'] = train_data['close'].ewm(span=span, adjust=False).mean()
C:\Users\DeLL\AppData\Local\Temp\ipykernel_1424\1934862544.py:38: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  test_data['ema'] = test_data['close'].ewm(span=span, adjust=False).mean()
```


```
MSE error for EMA on Test Data: 40.60396
```

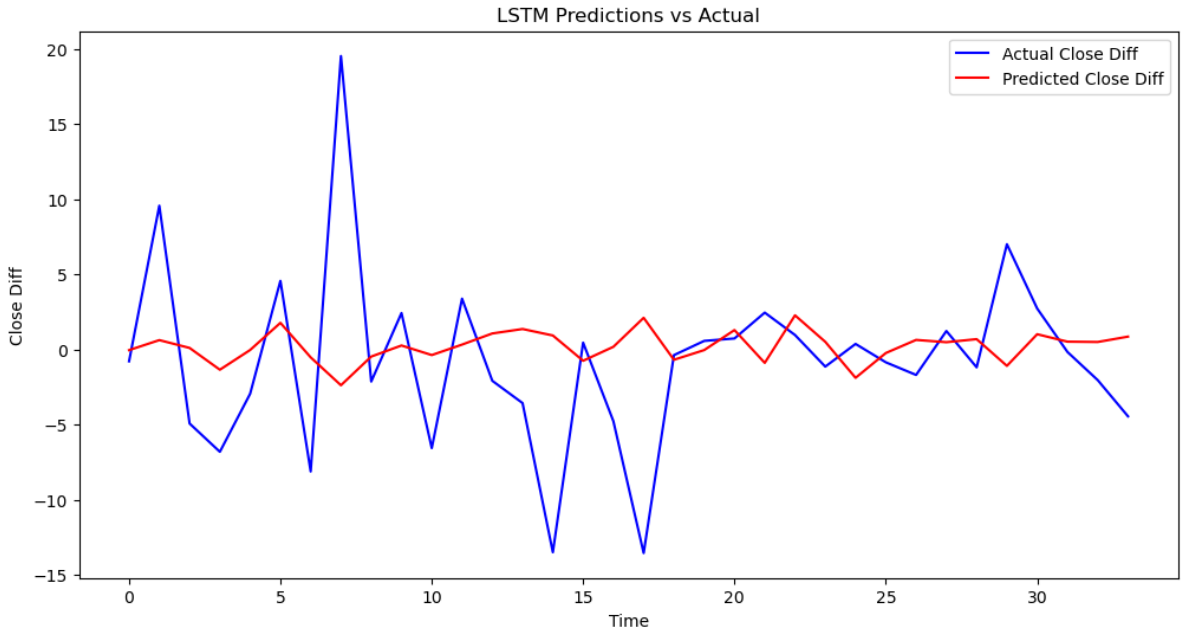
LSTM

In [206]:

```
# Assuming ibm_data is your DataFrame and 'close_diff' is the target column
data = ibm_data['close_diff'].values

# Function to create sequences for LSTM
def create_sequences(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:(i + window_size)])
        y.append(data[i + window_size])
    return np.array(X), np.array(y)

# Create sequences with a window size of 12
window_size = 10
X, y = create_sequences(data, window_size)

# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, ran

# Scaling the data with separate scalers
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))

# Reshape input to be [samples, time steps, features]
X_train_scaled = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_sc
X_test_scaled = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled

# Building the LSTM model
model = Sequential()
model.add(LSTM(90, return_sequences=True, input_shape=(window_size, 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(50))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
early_stopping = EarlyStopping(monitor='val_loss', patience=10, mode='min')
model.fit(X_train_scaled, y_train_scaled, validation_data=(X_test_scaled, y_

# Making predictions
predictions_scaled = model.predict(X_test_scaled)

# Inverse transform predictions
predictions_inverse = scaler_y.inverse_transform(predictions_scaled)

# Inverse transform test labels
y_test_inverse = scaler_y.inverse_transform(y_test_scaled)

# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(y_test_inverse, label='Actual Close Diff', color='blue')
plt.plot(predictions_inverse, label='Predicted Close Diff', color='red')
plt.title('LSTM Predictions vs Actual')
plt.xlabel('Time')
plt.ylabel('Close Diff')
plt.legend()
plt.show()
```

```
64
65 #evealuation metric
66 rmse = np.sqrt(mean_squared_error(y_test_inverse, predictions_inverse))
67
68
69 mse = mean_squared_error(y_test_inverse, predictions_inverse)
70
71
72 mae = mean_absolute_error(y_test_inverse, predictions_inverse)
73
74 print(f"RMSE: {rmse}")
75 print(f"MSE: {mse}")
76 print(f"MAE: {mae}")
77
```

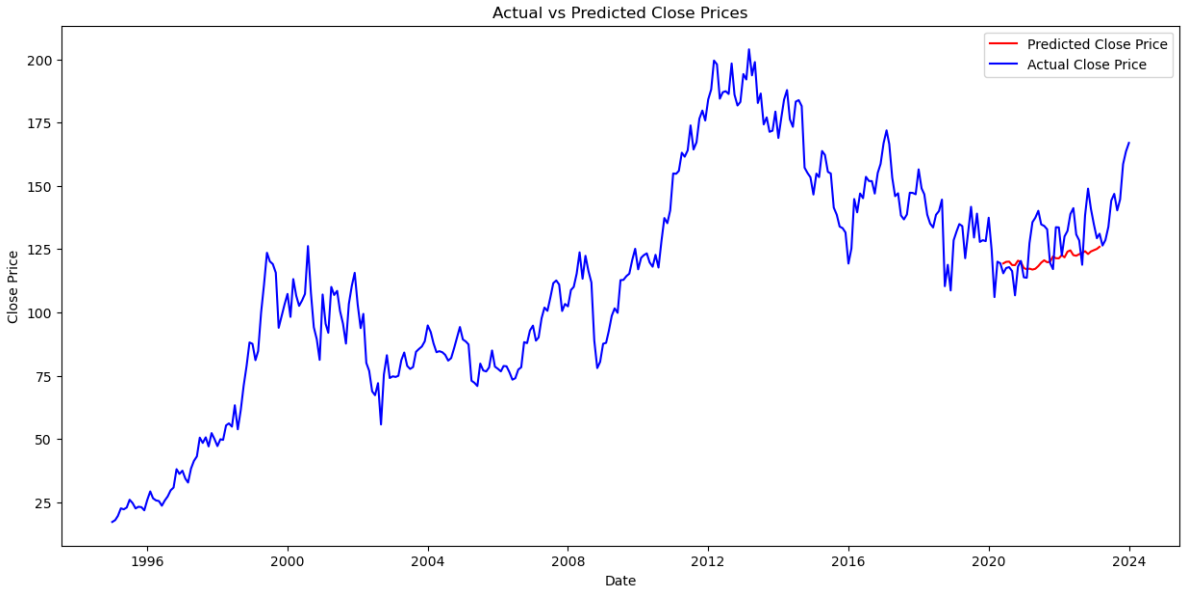
```
Epoch 1/20
6/6 [==============================] - 9s 258ms/step - loss: 0.1835 - val_loss:
0.0151
Epoch 2/20
6/6 [==============================] - 0s 23ms/step - loss: 0.0462 - val_loss:
0.0185
Epoch 3/20
6/6 [==============================] - 0s 18ms/step - loss: 0.0212 - val_loss:
0.0192
Epoch 4/20
6/6 [==============================] - 0s 15ms/step - loss: 0.0273 - val_loss:
0.0116
Epoch 5/20
6/6 [==============================] - 0s 18ms/step - loss: 0.0201 - val_loss:
0.0182
Epoch 6/20
6/6 [==============================] - 0s 17ms/step - loss: 0.0203 - val_loss:
0.0116
Epoch 7/20
6/6 [==============================] - 0s 16ms/step - loss: 0.0191 - val_loss:
0.0115
Epoch 8/20
6/6 [==============================] - 0s 18ms/step - loss: 0.0188 - val_loss:
0.0118
Epoch 9/20
6/6 [==============================] - 0s 18ms/step - loss: 0.0186 - val_loss:
0.0133
Epoch 10/20
6/6 [==============================] - 0s 24ms/step - loss: 0.0184 - val_loss:
0.0113
Epoch 11/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0186 - val_loss:
0.0112
Epoch 12/20
6/6 [==============================] - 0s 17ms/step - loss: 0.0185 - val_loss:
0.0122
Epoch 13/20
6/6 [==============================] - 0s 17ms/step - loss: 0.0184 - val_loss:
0.0115
Epoch 14/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0183 - val_loss:
0.0115
Epoch 15/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0184 - val_loss:
0.0113
Epoch 16/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0187 - val_loss:
0.0122
Epoch 17/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0183 - val_loss:
0.0112
Epoch 18/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0185 - val_loss:
0.0115
Epoch 19/20
6/6 [==============================] - 0s 20ms/step - loss: 0.0184 - val_loss:
0.0118
Epoch 20/20
6/6 [==============================] - 0s 19ms/step - loss: 0.0183 - val_loss:
0.0114
2/2 [==============================] - 1s 6ms/step
```




```
RMSE: 6.401553081709281
MSE: 40.97988185794159
MAE: 4.31596127853674
```

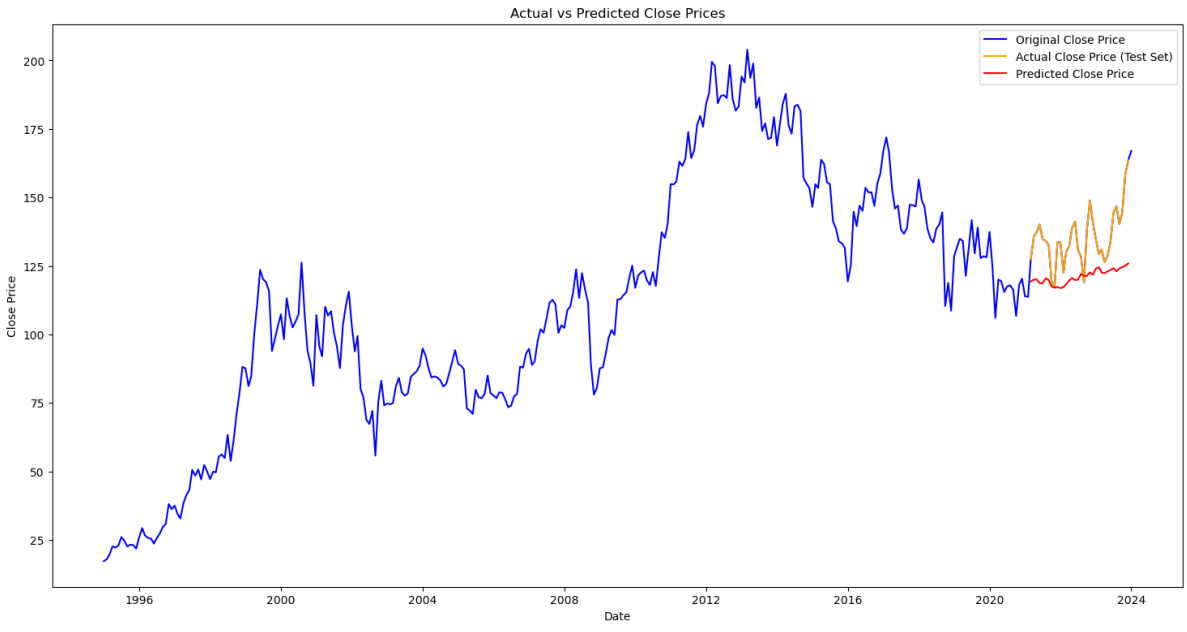
In [207]:

```
# Define the split_index
split_index = len(X_train)

# Calculate the last actual 'close' price from the training set
last_actual_close = ibm_data.iloc[split_index - 1]['close']

# Reconstruct the predicted 'close' prices by cumulatively summing the predi
predicted_close_prices = last_actual_close + np.cumsum(predictions_inverse.f

# Plot the actual close prices and the predicted close prices
plt.figure(figsize=(15, 7))
plt.plot(ibm_data.index[split_index:split_index + len(predicted_close_prices
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Actual Clos
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

```



In [208]:

```
# Define the split_index
split_index = len(X_train)

# Assuming ibm_data contains the original 'close' column
# and the last value of the training data's 'close' column as the starting p
last_train_close = ibm_data.iloc[split_index - 1]['close']

# Reconstruct the predicted close prices
# Start with the last actual close price and add the cumulative sum of the pr
reconstructed_predictions = last_train_close + np.cumsum(predictions_inverse

# Extract the actual close prices corresponding to the test data
actual_close_prices = ibm_data['close'][split_index + window_size - 1: split_

# Ensure that predicted_dates and reconstructed_predictions have the same le
predicted_dates = ibm_data.index[split_index + window_size - 1: split_index

# Plotting the original close data and the predicted close prices
plt.figure(figsize=(18, 9))
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Original Cl
plt.plot(predicted_dates, actual_close_prices, color='orange', label='Actual
plt.plot(predicted_dates, reconstructed_predictions, color='red', label='Pre
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

```



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

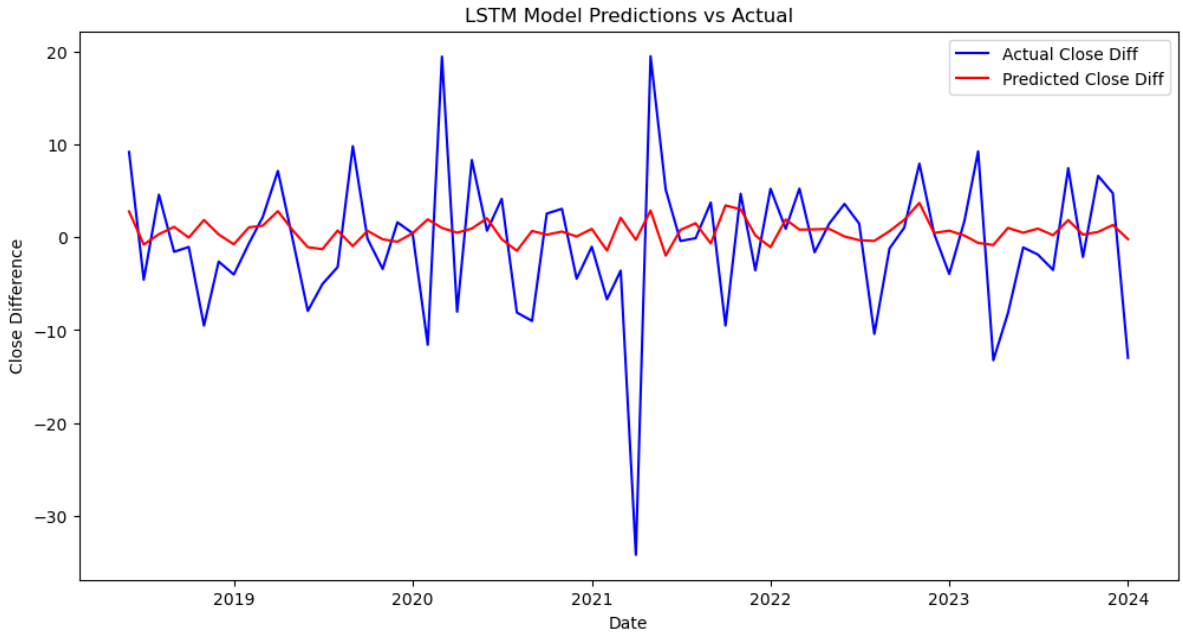
Forecast Horizons t+1 ,t+3 ,t+6

In [214]:

```
# Differencing the 'close' column
ibm_data['close_diff'] = ibm_data['close'].diff().fillna(0)

# Create sequences
window_size = 12
X, y = create_sequences(ibm_data['close_diff'].values, window_size, 1)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Scale the data
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))

# Reshape for LSTM
X_train_scaled = X_train_scaled.reshape((X_train_scaled.shape[0], window_siz
X_test_scaled = X_test_scaled.reshape((X_test_scaled.shape[0], window_size,

# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(window_size, 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train_scaled, y_train_scaled, epochs=100, batch_size=32, validat

# Make predictions
predictions = scaler_y.inverse_transform(model.predict(X_test_scaled))

# Calculate RMSE, MSE, MAE
rmse = np.sqrt(mean_squared_error(y_test, predictions))
mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(ibm_data.index[-len(y_test):], y_test, label='Actual Close Diff', c
plt.plot(ibm_data.index[-len(predictions):], predictions.flatten(), label='P
plt.title('LSTM Model Predictions vs Actual')
plt.xlabel('Date')
plt.ylabel('Close Difference')
plt.legend()
plt.show()

print(f'RMSE: {rmse:.5f}')
print(f'MSE: {mse:.5f}')
print(f'MAE: {mae:.5f}')

```

```
Epoch 1/100
9/9 [==============================] - 5s 120ms/step - loss: 0.1174 - val_loss:
0.0323
Epoch 2/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0383 - val_loss:
0.0234
Epoch 3/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0279 - val_loss:
0.0256
Epoch 4/100
9/9 [==============================] - 0s 25ms/step - loss: 0.0252 - val_loss:
0.0262
Epoch 5/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0242 - val_loss:
0.0238
Epoch 6/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0236 - val_loss:
0.0227
Epoch 7/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0228 - val_loss:
0.0243
Epoch 8/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0233 - val_loss:
0.0235
Epoch 9/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0231 - val_loss:
0.0227
Epoch 10/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0227 - val_loss:
0.0240
Epoch 11/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0230 - val_loss:
0.0235
Epoch 12/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0227 - val_loss:
0.0226
Epoch 13/100
9/9 [==============================] - 0s 13ms/step - loss: 0.0227 - val_loss:
0.0242
Epoch 14/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0232 - val_loss:
0.0228
Epoch 15/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0228 - val_loss:
0.0234
Epoch 16/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0227 - val_loss:
0.0230
Epoch 17/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0227 - val_loss:
0.0230
Epoch 18/100
9/9 [==============================] - 0s 13ms/step - loss: 0.0230 - val_loss:
0.0233
Epoch 19/100
9/9 [==============================] - 0s 13ms/step - loss: 0.0226 - val_loss:
0.0228
Epoch 20/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0225 - val_loss:
0.0229
Epoch 21/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0225 - val_loss:
0.0227
```

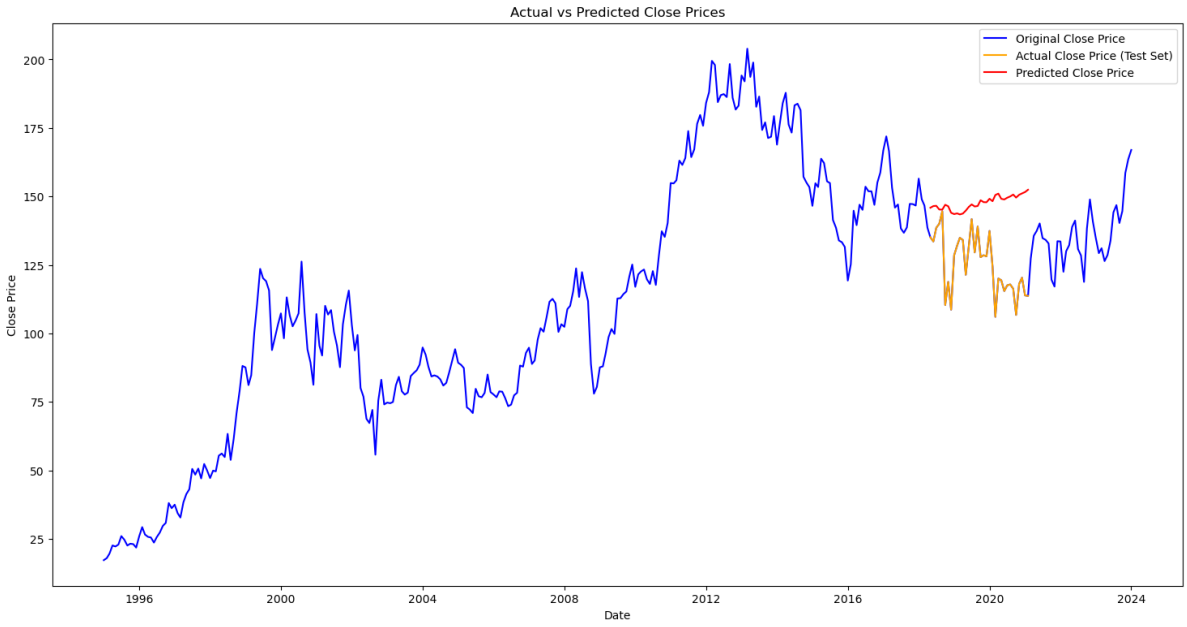
```
Epoch 22/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0225 - val_loss:
0.0228
3/3 [==============================] - 1s 5ms/step
```




```
RMSE: 7.56472
MSE: 57.22504
MAE: 5.34752
```

In [215]:

```
# Define the split_index
split_index = len(X_train)

# and the last value of the training data's 'close' column as the starting p
last_train_close = ibm_data.iloc[split_index - 1]['close']

# Reconstruct the predicted close prices
# Start with the last actual close price and add the cumulative sum of the p
reconstructed_predictions = last_train_close + np.cumsum(predictions_inverse

# Extract the actual close prices corresponding to the test data
actual_close_prices = ibm_data['close'][split_index + window_size - 1: split

# Ensure that predicted_dates and reconstructed_predictions have the same le
predicted_dates = ibm_data.index[split_index + window_size - 1: split_index

# Plotting the original close data and the predicted close prices
plt.figure(figsize=(18, 9))
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Original Cl
plt.plot(predicted_dates, actual_close_prices, color='orange', label='Actual
plt.plot(predicted_dates, reconstructed_predictions, color='red', label='Pre
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

```



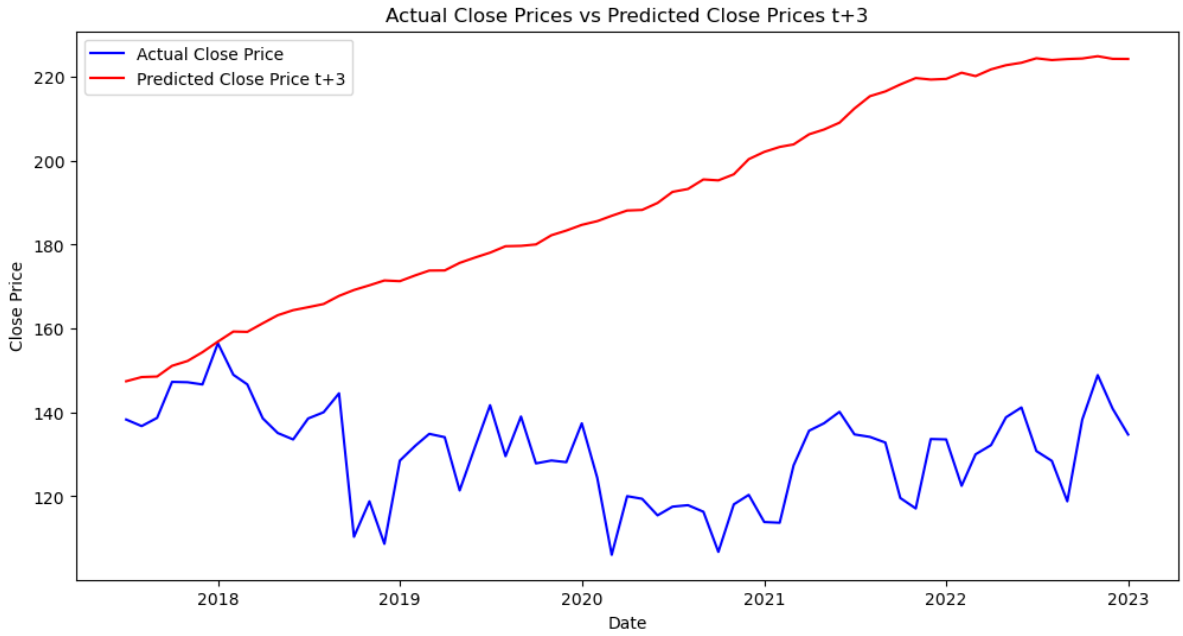
t+3

In [115]:

```
# Update forecast horizon for t+3
forecast_horizon_t3 = 3
X_t3, y_t3 = create_sequences(data, window_size, forecast_horizon_t3)

# Splitting the data into train and test sets for t+3
X_train_t3, X_test_t3, y_train_t3, y_test_t3 = train_test_split(X_t3, y_t3,

# Scaling the data for t+3
X_train_t3_scaled = scaler_X.fit_transform(X_train_t3.reshape(-1, 1))
X_train_t3_scaled = X_train_t3_scaled.reshape((X_train_t3.shape[0], window_s
X_test_t3_scaled = scaler_X.transform(X_test_t3.reshape(-1, 1))
X_test_t3_scaled = X_test_t3_scaled.reshape((X_test_t3.shape[0], window_size

y_train_t3_scaled = scaler_y.fit_transform(y_train_t3.reshape(-1, 1))
y_test_t3_scaled = scaler_y.transform(y_test_t3.reshape(-1, 1))

# Building the LSTM model for t+3 forecasting
model_t3 = Sequential()
model_t3.add(LSTM(50, return_sequences=True, input_shape=(window_size, 1)))
model_t3.add(LSTM(50))
model_t3.add(Dense(1))

model_t3.compile(optimizer='adam', loss='mean_squared_error')

# Training the model with early stopping for t+3
early_stopping_t3 = EarlyStopping(monitor='val_loss', patience=10, mode='min
model_t3.fit(X_train_t3_scaled, y_train_t3_scaled, epochs=100, batch_size=32

# Making predictions for t+3
predictions_scaled_t3 = model_t3.predict(X_test_t3_scaled)

# Inverse transforming the predictions for t+3
predictions_diff_t3 = scaler_y.inverse_transform(predictions_scaled_t3)

# Reverse the differencing to get the predicted 'close' prices for t+3
predicted_close_prices_t3 = [last_train_close + np.sum(predictions_diff_t3[:

# Get the corresponding actual 'close' prices for the test set for t+3
actual_close_prices_t3 = ibm_data['close'][len(y_train_t3) + forecast_horizo

# Plotting the results for t+3
plt.figure(figsize=(12, 6))
plt.plot(ibm_data.index[len(y_train_t3) + forecast_horizon_t3 - 1: len(y_tra
plt.plot(ibm_data.index[len(y_train_t3) + forecast_horizon_t3 - 1: len(y_tra
plt.title('Actual Close Prices vs Predicted Close Prices t+3')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

#Ensure that the lengths of y_test_t3 and predictions_diff_t3 match
min_length_t3 = min(len(y_test_t3), len(predictions_diff_t3))

#Trim the longer array to match the length of the shorter one
y_test_t3_aligned = y_test_t3[:min_length_t3]
predictions_t3_aligned = predictions_diff_t3[:min_length_t3]

#Calculate evaluation metrics for t+3
rmse_t3 = np.sqrt(mean_squared_error(y_test_t3_aligned, predictions_t3_align
mse_t3 = mean_squared_error(y_test_t3_aligned, predictions_t3_aligned)
mae_t3 = mean_absolute_error(y_test_t3_aligned, predictions_t3_aligned)

print(f'RMSE t+3: {rmse_t3}')
```

```
print(f'MSE t+3: {mse_t3}')
print(f'MAE t+3: {mae_t3}')

```

```
Epoch 1/100
9/9 [==============================] - 5s 140ms/step - loss: 0.1040 - val_loss:
0.0565
Epoch 2/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0404 - val_loss:
0.0271
Epoch 3/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0273 - val_loss:
0.0270
Epoch 4/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0249 - val_loss:
0.0305
Epoch 5/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0242 - val_loss:
0.0271
Epoch 6/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0242 - val_loss:
0.0277
Epoch 7/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0242 - val_loss:
0.0281
Epoch 8/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0242 - val_loss:
0.0277
Epoch 9/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0240 - val_loss:
0.0282
Epoch 10/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0243 - val_loss:
0.0276
Epoch 11/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0240 - val_loss:
0.0278
Epoch 12/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0240 - val_loss:
0.0276
Epoch 13/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0241 - val_loss:
0.0284
3/3 [==============================] - 1s 6ms/step
```




```
RMSE t+3: 8.452435085759785
MSE t+3: 71.44365887898303
MAE t+3: 5.960754663447168
```

t+6

In [116]:

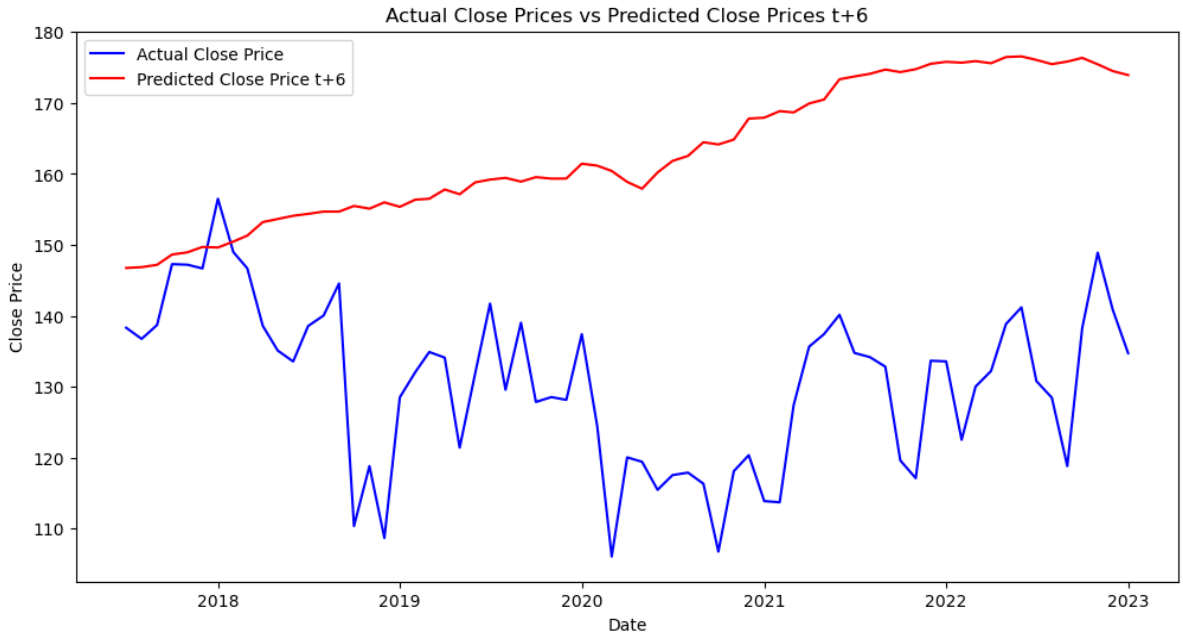
```
# Update forecast horizon for t+6
forecast_horizon_t6 = 6
X_t6, y_t6 = create_sequences(data, window_size, forecast_horizon_t6)

# Splitting the data into train and test sets for t+6
X_train_t6, X_test_t6, y_train_t6, y_test_t6 = train_test_split(X_t6, y_t6,

# Scaling the data for t+6
X_train_t6_scaled = scaler_X.fit_transform(X_train_t6.reshape(-1, 1))
X_train_t6_scaled = X_train_t6_scaled.reshape((X_train_t6.shape[0], window_s
X_test_t6_scaled = scaler_X.transform(X_test_t6.reshape(-1, 1))
X_test_t6_scaled = X_test_t6_scaled.reshape((X_test_t6.shape[0], window_size

y_train_t6_scaled = scaler_y.fit_transform(y_train_t6.reshape(-1, 1))
y_test_t6_scaled = scaler_y.transform(y_test_t6.reshape(-1, 1))

# Building the LSTM model for t+6 forecasting
model_t6 = Sequential()
model_t6.add(LSTM(50, return_sequences=True, input_shape=(window_size, 1)))
model_t6.add(LSTM(50))
model_t6.add(Dense(1))

model_t6.compile(optimizer='adam', loss='mean_squared_error')

# Training the model with early stopping for t+6
early_stopping_t6 = EarlyStopping(monitor='val_loss', patience=10, mode='min
model_t6.fit(X_train_t6_scaled, y_train_t6_scaled, epochs=100, batch_size=32

# Making predictions for t+6
predictions_scaled_t6 = model_t6.predict(X_test_t6_scaled)

# Inverse transforming the predictions for t+6
predictions_diff_t6 = scaler_y.inverse_transform(predictions_scaled_t6)

# Reverse the differencing to get the predicted 'close' prices for t+6
predicted_close_prices_t6 = [last_train_close + np.sum(predictions_diff_t6[:

# Get the corresponding actual 'close' prices for the test set for t+6

actual_close_prices_t6 = ibm_data['close'][len(y_train_t6) + forecast_horizo

#Plotting the results for t+6
plt.figure(figsize=(12, 6))
plt.plot(ibm_data.index[len(y_train_t6) + forecast_horizon_t6 - 1: len(y_tra
plt.plot(ibm_data.index[len(y_train_t6) + forecast_horizon_t6 - 1: len(y_tra
plt.title('Actual Close Prices vs Predicted Close Prices t+6')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

#Ensure that the lengths of y_test_t6 and predictions_diff_t6 match
min_length_t6 = min(len(y_test_t6), len(predictions_diff_t6))

#Trim the longer array to match the length of the shorter one
y_test_t6_aligned = y_test_t6[:min_length_t6]
predictions_t6_aligned = predictions_diff_t6[:min_length_t6]

#Calculate evaluation metrics for t+6
rmse_t6 = np.sqrt(mean_squared_error(y_test_t6_aligned, predictions_t6_align
mse_t6 = mean_squared_error(y_test_t6_aligned, predictions_t6_aligned)
mae_t6 = mean_absolute_error(y_test_t6_aligned, predictions_t6_aligned)
```

```
64
65 print(f'RMSE t+6: {rmse_t6}')
66 print(f'MSE t+6: {mse_t6}')
67 print(f'MAE t+6: {mae_t6}')
```

```
Epoch 1/100
9/9 [==============================] - 5s 137ms/step - loss: 0.3109 - val_loss:
0.0258
Epoch 2/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0491 - val_loss:
0.0297
Epoch 3/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0260 - val_loss:
0.0334
Epoch 4/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0290 - val_loss:
0.0214
Epoch 5/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0229 - val_loss:
0.0213
Epoch 6/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0226 - val_loss:
0.0214
Epoch 7/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0224 - val_loss:
0.0208
Epoch 8/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0226 - val_loss:
0.0205
Epoch 9/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0221 - val_loss:
0.0209
Epoch 10/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0221 - val_loss:
0.0204
Epoch 11/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0228 - val_loss:
0.0205
Epoch 12/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0233 - val_loss:
0.0220
Epoch 13/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0225 - val_loss:
0.0206
Epoch 14/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0222 - val_loss:
0.0212
Epoch 15/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0222 - val_loss:
0.0205
Epoch 16/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0222 - val_loss:
0.0204
Epoch 17/100
9/9 [==============================] - 0s 20ms/step - loss: 0.0221 - val_loss:
0.0210
Epoch 18/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0221 - val_loss:
0.0204
Epoch 19/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0222 - val_loss:
0.0207
Epoch 20/100
9/9 [==============================] - 0s 18ms/step - loss: 0.0222 - val_loss:
0.0205
Epoch 21/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0222 - val_loss:
0.0205
```

```
Epoch 22/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0223 - val_loss:
0.0204
Epoch 23/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0228 - val_loss:
0.0206
Epoch 24/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0227 - val_loss:
0.0204
Epoch 25/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0234 - val_loss:
0.0212
Epoch 26/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0230 - val_loss:
0.0204
Epoch 27/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0224 - val_loss:
0.0221
Epoch 28/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0222 - val_loss:
0.0205
Epoch 29/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0222 - val_loss:
0.0208
Epoch 30/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0224 - val_loss:
0.0206
Epoch 31/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0221 - val_loss:
0.0210
Epoch 32/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0221 - val_loss:
0.0203
Epoch 33/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0225 - val_loss:
0.0205
Epoch 34/100
9/9 [==============================] - 0s 17ms/step - loss: 0.0224 - val_loss:
0.0205
Epoch 35/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0220 - val_loss:
0.0211
Epoch 36/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0222 - val_loss:
0.0205
Epoch 37/100
9/9 [==============================] - 0s 13ms/step - loss: 0.0223 - val_loss:
0.0207
Epoch 38/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0220 - val_loss:
0.0211
Epoch 39/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0220 - val_loss:
0.0203
Epoch 40/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0224 - val_loss:
0.0208
Epoch 41/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0220 - val_loss:
0.0203
Epoch 42/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0219 - val_loss:
0.0206
```

```
Epoch 43/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0220 - val_loss:
0.0205
Epoch 44/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0221 - val_loss:
0.0204
Epoch 45/100
9/9 [==============================] - 0s 14ms/step - loss: 0.0222 - val_loss:
0.0205
Epoch 46/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0233 - val_loss:
0.0204
Epoch 47/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0226 - val_loss:
0.0205
Epoch 48/100
9/9 [==============================] - 0s 16ms/step - loss: 0.0220 - val_loss:
0.0206
Epoch 49/100
9/9 [==============================] - 0s 15ms/step - loss: 0.0223 - val_loss:
0.0204
3/3 [==============================] - 1s 5ms/step
```




```
RMSE t+6: 7.677719068179569
MSE t+6: 58.94737008988816
MAE t+6: 5.742031194276607
```

Multivariate lstm model

INDUSTRIAL PRODUCTION INDEX

In [133]:

```
# Load the data
ipi_index = pd.read_csv('F:/data/Industrial production index.csv')
ibm_data = ibm_data

# Preprocess the Industrial Production Index
ipi_index['DATE'] = pd.to_datetime(ipi_index['DATE']).dt.to_period('M')
ipi_index.set_index('DATE', inplace=True)
ipi_index.index = ipi_index.index.to_timestamp()

# Filter out rows where the index (DATE) is on or after January 1995
ipi_index = ipi_index[ipi_index.index >= pd.to_datetime('1995-01-01')]

# Calculate the first difference for both datasets
ibm_data['close_diff'] = ibm_data['close'].diff()
ipi_index['IPI_diff'] = ipi_index['INDPRO'].diff()

# Drop the first row with NaN values after differencing
ibm_data = ibm_data.dropna()
ipi_index = ipi_index.dropna()

# Merge the dataframes on the DATE column
merged_data = pd.merge(ibm_data, ipi_index, left_index=True, right_index=Tru

#Convert the index to datetime, if it's not already
merged_data.index = pd.to_datetime(merged_data.index, infer_datetime_format=

#Select the differenced features and target
target = merged_data['close_diff'].values
features = merged_data[['IPI_diff']].values # Add other differenced features

#Initialize the scaler
scaler_x = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

#Scale the features and target
features_scaled = scaler_x.fit_transform(features)
target_scaled = scaler_y.fit_transform(target.reshape(-1, 1)).flatten()

#Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target_

#Reshape the data for LSTM layer
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#Build the LSTM model
model = Sequential()
model.add(LSTM(units=60, return_sequences=False, input_shape=(X_train.shape[
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

#Train the model
model.fit(X_train, y_train, epochs=100, batch_size=24, verbose=1)

#Predict and inverse transform to original scale
predictions_scaled = model.predict(X_test)
predictions = scaler_y.inverse_transform(predictions_scaled)

#Inverse transform y_test
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten(

#Calculate MSE and RMSE
mse = mean_squared_error(y_test_original, predictions)
```

```
rmse = sqrt(mse)
mae = mean_absolute_error(y_test_original, predictions)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:",mae)
#Extract the dates from merged_data for the test set for plotting
test_dates = merged_data.index[-len(predictions):]

#Plotting
plt.figure(figsize=(18, 9))
plt.plot(test_dates, y_test_original, label='Actual Differenced Close Price'
plt.plot(test_dates, predictions, label='Predicted Differenced Close Price')
plt.title('Actual vs Predicted Differenced Close Prices')
plt.xlabel('Date')
plt.ylabel('Differenced Close Price')
plt.legend()
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.show()
```

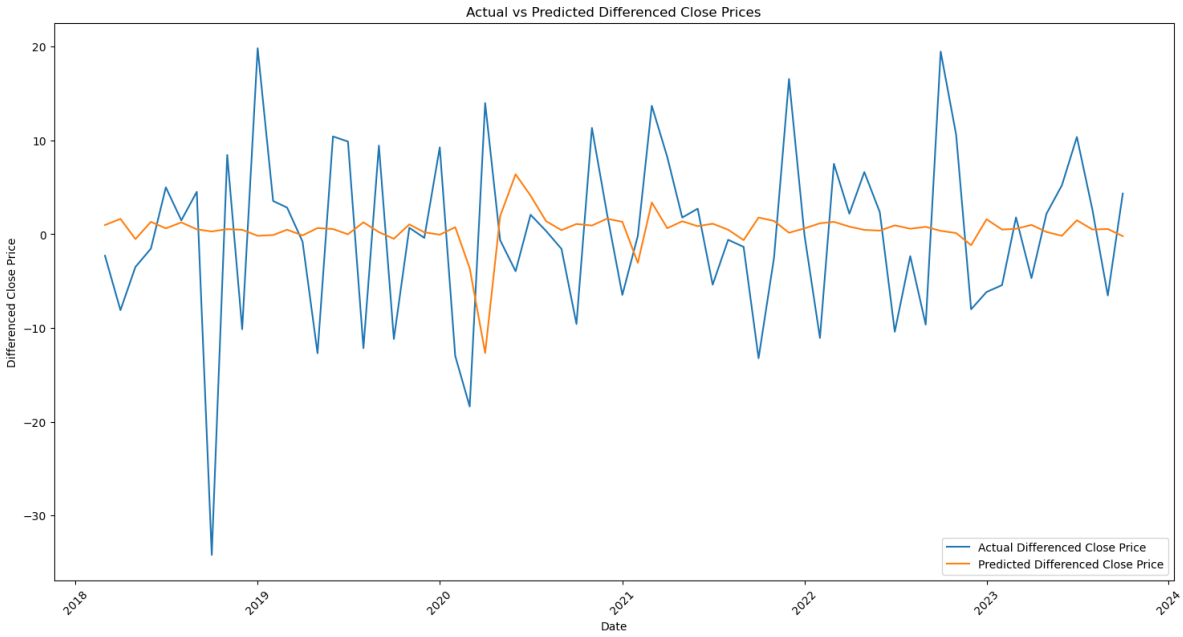
```
C:\Users\DeLL\AppData\Local\Temp\ipykernel_16588\1698474600.py:34: UserWarning:
The argument 'infer_datetime_format' is deprecated and will be removed in a fut
ure version. A strict version of it is now the default, see https://pandas.pyda
ta.org/pdeps/0004-consistent-to-datetime-parsing.html. (https://pandas.pydata.o
rg/pdeps/0004-consistent-to-datetime-parsing.html.) You can safely remove this
argument.
  merged_data.index = pd.to_datetime(merged_data.index, infer_datetime_format=T
rue)
```

```
Epoch 1/100
12/12 [==============================] - 2s 3ms/step - loss: 0.3377
Epoch 2/100
12/12 [==============================] - 0s 3ms/step - loss: 0.2608
Epoch 3/100
12/12 [==============================] - 0s 4ms/step - loss: 0.1945
Epoch 4/100
12/12 [==============================] - 0s 3ms/step - loss: 0.1367
Epoch 5/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0898
Epoch 6/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0547
Epoch 7/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0318
Epoch 8/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0202
Epoch 9/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 10/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 11/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 12/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 13/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 14/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 15/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 16/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 17/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 18/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 19/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 20/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 21/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 22/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 23/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 24/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 25/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 26/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 27/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 28/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 29/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 30/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 31/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 32/100
```

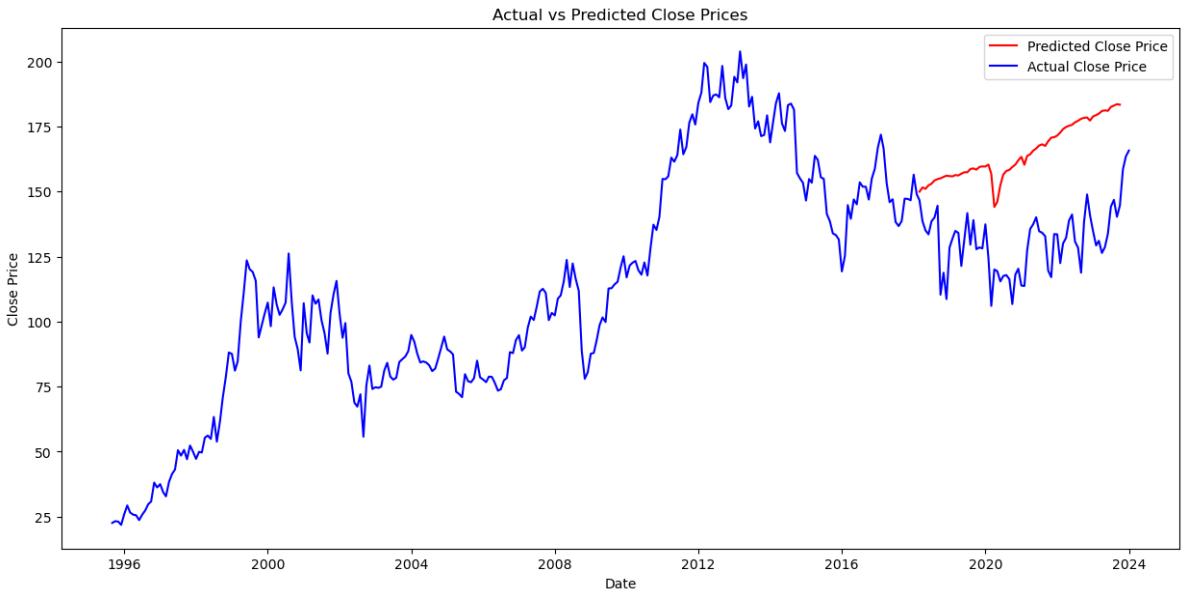
```
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 33/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 34/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 35/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 36/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 37/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 38/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 39/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 40/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 41/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 42/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 43/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 44/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 45/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 46/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 47/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 48/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 49/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 50/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 51/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 52/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 53/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 54/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 55/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 56/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 57/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0153
Epoch 58/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 59/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 60/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 61/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 62/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 63/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
```

```
Epoch 64/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 65/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 66/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 67/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0154
Epoch 68/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 69/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 70/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 71/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 72/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 73/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 74/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 75/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 76/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 77/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 78/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0153
Epoch 79/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 80/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 81/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 82/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 83/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 84/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 85/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 86/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 87/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 88/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 89/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 90/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 91/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 92/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 93/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0153
Epoch 94/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 95/100
```

```
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 96/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 97/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 98/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 99/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 100/100
12/12 [==============================] - 0s 2ms/step - loss: 0.0154
3/3 [==============================] - 0s 2ms/step
Mean Squared Error: 93.40607190351841
Root Mean Squared Error: 9.664681676264273
Mean Absolute Error: 7.256883904118748
```



In [134]:
```
# Start with the last actual close price from the training set
last_train_close = ibm_data.iloc[len(X_train) - 1]['close']

# Reconstruct the predicted 'close' prices
# The predictions are the differenced values, so we need to add these to the
predicted_close_prices = [last_train_close]
for pred_diff in predictions.flatten():
    predicted_close_prices.append(predicted_close_prices[-1] + pred_diff)

# Remove the first element which was the last actual close price
predicted_close_prices.pop(0)

# Extract the test set dates
test_dates = merged_data.index[len(X_train):len(X_train) + len(predicted_clo

# Plotting the actual and predicted close prices
plt.figure(figsize=(15, 7))
plt.plot(test_dates, predicted_close_prices, color='red', label='Predicted C
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Actual Clos
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

```



In [ ]:
```
1
```

In [ ]:
```
df_input.to_csv('C:/Users/DeLL/Documents/ibm_data and IPI index.csv', index=
```

Non Farm payroll employment

In [155]:

```
# Load the Non-Farm Payroll Employment data
payems = pd.read_csv('F:/data/Non_farm_payroll_employment.csv')

# Preprocess the Non-Farm Payroll Employment data
payems['DATE'] = pd.to_datetime(payems['DATE'])
payems['DATE'] = payems['DATE'].dt.strftime('%Y-%m')
payems.set_index('DATE', inplace=True)

# Ensure the index is in datetime format for comparison
payems.index = pd.to_datetime(payems.index, format='%Y-%m')  # Adjust the fo

# Filter out rows where the index (DATE) is on or after January 1995
payems = payems[payems.index >= pd.to_datetime('1995-01')]

# Calculate the first difference for the payems dataset
payems['PAYEMS_diff'] = payems['PAYEMS'].diff()

# Drop the first row with NaN values after differencing
payems = payems.dropna()

# Merge the differenced payems data with the ibm_data (assuming ibm_data has
merged_data = pd.merge(ibm_data, payems, left_index=True, right_index=True,

# Select the differenced features and target
target = merged_data['close_diff'].values
features = merged_data[['PAYEMS_diff']].values  # Add other differenced feat

#Initialize the scaler
scaler_x = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

#Scale the features and target
features_scaled = scaler_x.fit_transform(features)
target_scaled = scaler_y.fit_transform(target.reshape(-1, 1)).flatten()

#Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target_

#Reshape the data for LSTM layer
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#Build the LSTM model
model = Sequential()
model.add(LSTM(units=30, return_sequences=False, input_shape=(X_train.shape[
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

#Train the model
model.fit(X_train, y_train, epochs=100, batch_size=12, verbose=1)

#Predict and inverse transform to original scale
predictions_scaled = model.predict(X_test)
predictions = scaler_y.inverse_transform(predictions_scaled)

#Inverse transform y_test
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten(

#Calculate MSE and RMSE
mse = mean_squared_error(y_test_original, predictions)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test_original, predictions)
```

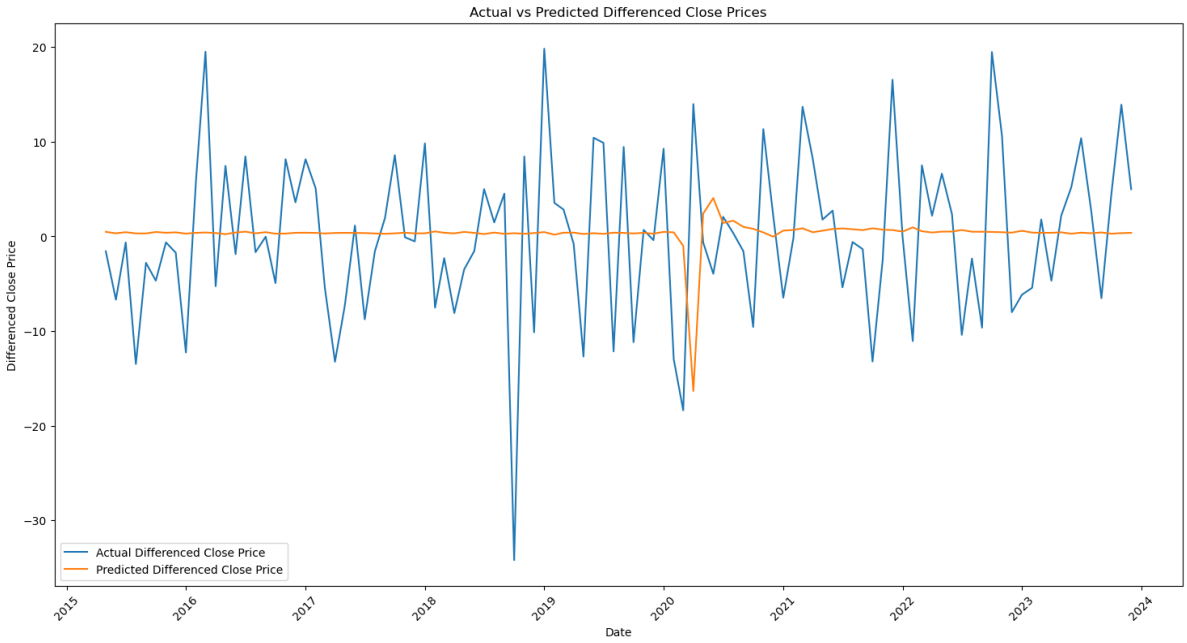
```
print("Mean Absolute Error:",mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)

#Extract the dates from merged_data for the test set for plotting
test_dates = merged_data.index[-len(predictions):]

#Plotting
plt.figure(figsize=(18, 9))
plt.plot(test_dates, y_test_original, label='Actual Differenced Close Price'
plt.plot(test_dates, predictions, label='Predicted Differenced Close Price')
plt.title('Actual vs Predicted Differenced Close Prices')
plt.xlabel('Date')
plt.ylabel('Differenced Close Price')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

```
Epoch 1/100
20/20 [==============================] - 2s 3ms/step - loss: 0.3012
Epoch 2/100
20/20 [==============================] - 0s 3ms/step - loss: 0.2015
Epoch 3/100
20/20 [==============================] - 0s 3ms/step - loss: 0.1228
Epoch 4/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0654
Epoch 5/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0326
Epoch 6/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0191
Epoch 7/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0156
Epoch 8/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 9/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 10/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 11/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 12/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 13/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 14/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 15/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 16/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 17/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 18/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 19/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 20/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 21/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 22/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 23/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 24/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 25/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 26/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 27/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 28/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 29/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 30/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 31/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 32/100
```

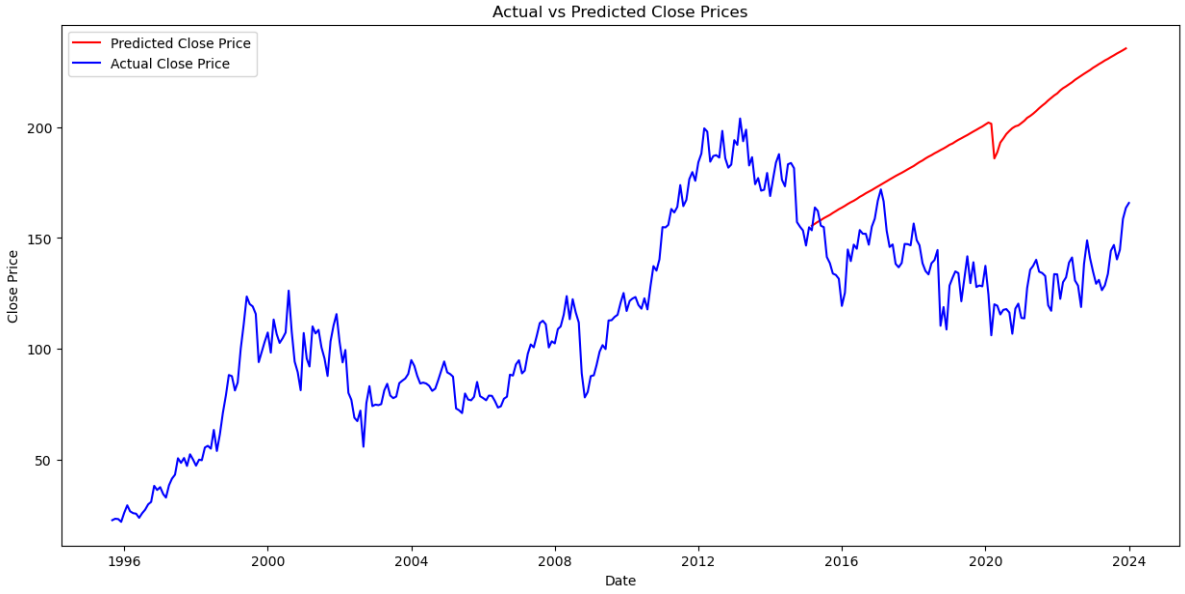
```
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 33/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 34/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 35/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 36/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0153
Epoch 37/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 38/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 39/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 40/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 41/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 42/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0156
Epoch 43/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 44/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 45/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 46/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 47/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 48/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 49/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 50/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 51/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 52/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 53/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 54/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 55/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 56/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 57/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 58/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 59/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 60/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 61/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 62/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 63/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
```

```
Epoch 64/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 65/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 66/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 67/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 68/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 69/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 70/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 71/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 72/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 73/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 74/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 75/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 76/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0156
Epoch 77/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 78/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0156
Epoch 79/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 80/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 81/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 82/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 83/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 84/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 85/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 86/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 87/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 88/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 89/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 90/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 91/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 92/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 93/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 94/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 95/100
```

```
20/20 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 96/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 97/100
20/20 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 98/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 99/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 100/100
20/20 [==============================] - 0s 2ms/step - loss: 0.0154
4/4 [==============================] - 0s 2ms/step
Mean Absolute Error: 6.823694143247289
Mean Squared Error: 81.77343287925186
Root Mean Squared Error: 9.042866408349283
```



In [138]:
```
# Start with the last actual close price from the training set
last_train_close = ibm_data.iloc[len(X_train) - 1]['close']

# Reconstruct the predicted 'close' prices
# The predictions are the differenced values, so we need to add these to the
predicted_close_prices = [last_train_close]
for pred_diff in predictions.flatten():
    predicted_close_prices.append(predicted_close_prices[-1] + pred_diff)

# Remove the first element which was the last actual close price
predicted_close_prices.pop(0)

# Extract the test set dates
test_dates = merged_data.index[len(X_train):len(X_train) + len(predicted_clo

# Plotting the actual and predicted close prices
plt.figure(figsize=(15, 7))
plt.plot(test_dates, predicted_close_prices, color='red', label='Predicted C
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Actual Clos
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



In [ ]:
```

```

In [ ]:
```
df_input.to_csv('C:/Users/DeLL/Documents/ibm_data and Non Farm payroll emplo
```

real manufacturing and trade sales

In [156]:

```
# Load the Real manufacturing and trade sales data
rmats = pd.read_csv('F:/data/Real manufacturing and trade sales.csv')

# Preprocess the dataset
rmats['DATE'] = pd.to_datetime(rmats['DATE'])
rmats.set_index('DATE', inplace=True)

# Filter out rows where the index (DATE) is on or after January 1995
rmats = rmats[rmats.index >= pd.to_datetime('1995-01')]

# Assuming ibm_data is your main dataset which contains 'close' prices
# Calculate the first difference for both datasets to make them stationary
ibm_data['close_diff'] = ibm_data['close'].diff()
rmats['RMATS_diff'] = rmats['CMRMTSPL'].diff()  # Replace 'RMATS' with the a

# Drop the first row with NaN values after differencing
ibm_data = ibm_data.dropna()
rmats = rmats.dropna()

# Merge the dataframes on the DATE column
merged_data = pd.merge(ibm_data, rmats, left_index=True, right_index=True, h

# Select the differenced features and target
target = merged_data['close_diff'].values
features = merged_data[['RMATS_diff']].values  # Add other differenced featu

# Initialize the scaler
scaler_x = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

# Scale the features and target
features_scaled = scaler_x.fit_transform(features)
target_scaled = scaler_y.fit_transform(target.reshape(-1, 1)).flatten()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled
, target_scaled, test_size=0.31, shuffle=False)

#Reshape the data for LSTM layer
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=False, input_shape=(X_train.shape[
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

#Train the model
model.fit(X_train, y_train, epochs=100, batch_size=12, verbose=1)

#Predict and inverse transform to original scale
predictions_scaled = model.predict(X_test)
predictions = scaler_y.inverse_transform(predictions_scaled)

#Inverse transform y_test
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten(

#Calculate MSE and RMSE
mse = mean_squared_error(y_test_original, predictions)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test_original, predictions)
```

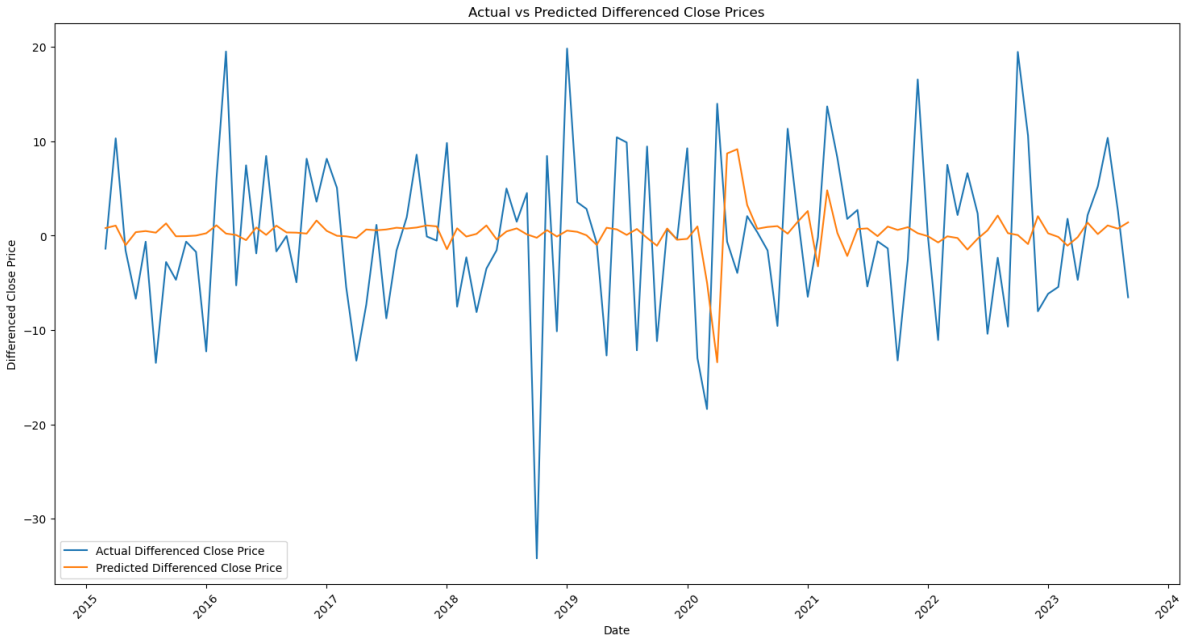
```
print("Mean Absolute Error:",mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)

#Extract the dates from merged_data for the test set for plotting
test_dates = merged_data.index[-len(predictions):]

#Plotting
plt.figure(figsize=(18, 9))
plt.plot(test_dates, y_test_original, label='Actual Differenced Close Price'
plt.plot(test_dates, predictions, label='Predicted Differenced Close Price')
plt.title('Actual vs Predicted Differenced Close Prices')
plt.xlabel('Date')
plt.ylabel('Differenced Close Price')
plt.legend()
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.show()
```

```
Epoch 1/100
19/19 [==============================] - 2s 3ms/step - loss: 0.2917
Epoch 2/100
19/19 [==============================] - 0s 2ms/step - loss: 0.1923
Epoch 3/100
19/19 [==============================] - 0s 2ms/step - loss: 0.1133
Epoch 4/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0561
Epoch 5/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0263
Epoch 6/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0165
Epoch 7/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 8/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 9/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 10/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 11/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 12/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 13/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 14/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 15/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 16/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 17/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 18/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 19/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 20/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 21/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 22/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 23/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 24/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 25/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 26/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 27/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 28/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 29/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 30/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 31/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 32/100
```

```
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 33/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 34/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 35/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 36/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 37/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 38/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 39/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 40/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 41/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 42/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 43/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 44/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 45/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 46/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 47/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 48/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 49/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 50/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 51/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 52/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 53/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 54/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 55/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 56/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 57/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 58/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 59/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 60/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 61/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 62/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 63/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
```

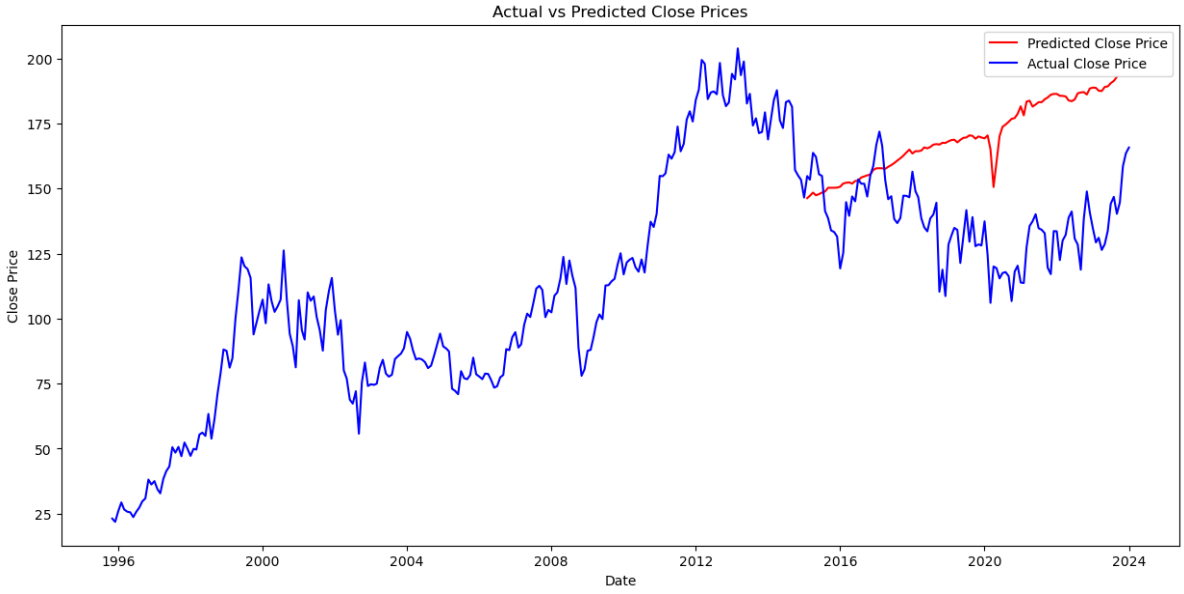
```
Epoch 64/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 65/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 66/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 67/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 68/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 69/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 70/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 71/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 72/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 73/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 74/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 75/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 76/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 77/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 78/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0155
Epoch 79/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 80/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 81/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 82/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 83/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 84/100
19/19 [==============================] - 0s 4ms/step - loss: 0.0155
Epoch 85/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 86/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 87/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 88/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 89/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 90/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 91/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 92/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0154
Epoch 93/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 94/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 95/100
```

```
19/19 [==============================] - 0s 2ms/step - loss: 0.0153
Epoch 96/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 97/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 98/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0155
Epoch 99/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0154
Epoch 100/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0156
4/4 [==============================] - 0s 3ms/step
Mean Absolute Error: 6.878750436957383
Mean Squared Error: 80.92844389238542
Root Mean Squared Error: 8.996023782337696
```



In [141]:

```
# Start with the last actual close price from the training set
last_train_close = ibm_data.iloc[len(X_train) - 1]['close']

# Reconstruct the predicted 'close' prices
# The predictions are the differenced values, so we need to add these to the
predicted_close_prices = [last_train_close]
for pred_diff in predictions.flatten():
    predicted_close_prices.append(predicted_close_prices[-1] + pred_diff)

# Remove the first element which was the last actual close price
predicted_close_prices.pop(0)

# Extract the test set dates
test_dates = merged_data.index[len(X_train):len(X_train) + len(predicted_clo

# Plotting the actual and predicted close prices
plt.figure(figsize=(15, 7))
plt.plot(test_dates, predicted_close_prices, color='red', label='Predicted C
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Actual Clos
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



In [ ]:

```
df_input.to_csv('C:/Users/DeLL/Documents/ibm_data and real manufacturing and
```

Real personal income excluding transfer receipt

In [158]:

```
# Load the Real manufacturing and trade sales data
rpietr = pd.read_csv('F:/data/Real personal income excluding transfer receip

# Preprocess the dataset
rpietr['DATE'] = pd.to_datetime(rpietr['DATE'])
rpietr.set_index('DATE', inplace=True)

# Filter out rows where the index (DATE) is on or after January 1995
rpietr = rpietr[rpietr.index >= pd.to_datetime('1995-01')]

# Assuming ibm_data is your main dataset which contains 'close' prices
# Calculate the first difference for both datasets to make them stationary
ibm_data['close_diff'] = ibm_data['close'].diff()
rpietr['rpietr_diff'] = rpietr['W875RX1'].diff()  # Replace 'RMATS' with the

# Drop the first row with NaN values after differencing
ibm_data = ibm_data.dropna()
rpietr = rpietr.dropna()

# Merge the dataframes on the DATE column
merged_data = pd.merge(ibm_data, rpietr, left_index=True, right_index=True,

# Select the differenced features and target
target = merged_data['close_diff'].values
features = merged_data[['rpietr_diff']].values  # Add other differenced feat

# Initialize the scaler
scaler_x = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

# Scale the features and target
features_scaled = scaler_x.fit_transform(features)
target_scaled = scaler_y.fit_transform(target.reshape(-1, 1)).flatten()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled
, target_scaled, test_size=0.31, shuffle=False)

#Reshape the data for LSTM layer
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=False, input_shape=(X_train.shape[
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

#Train the model
model.fit(X_train, y_train, epochs=100, batch_size=12, verbose=1)

#Predict and inverse transform to original scale
predictions_scaled = model.predict(X_test)
predictions = scaler_y.inverse_transform(predictions_scaled)

#Inverse transform y_test
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten(

#Calculate MSE and RMSE
mse = mean_squared_error(y_test_original, predictions)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test_original, predictions)
```

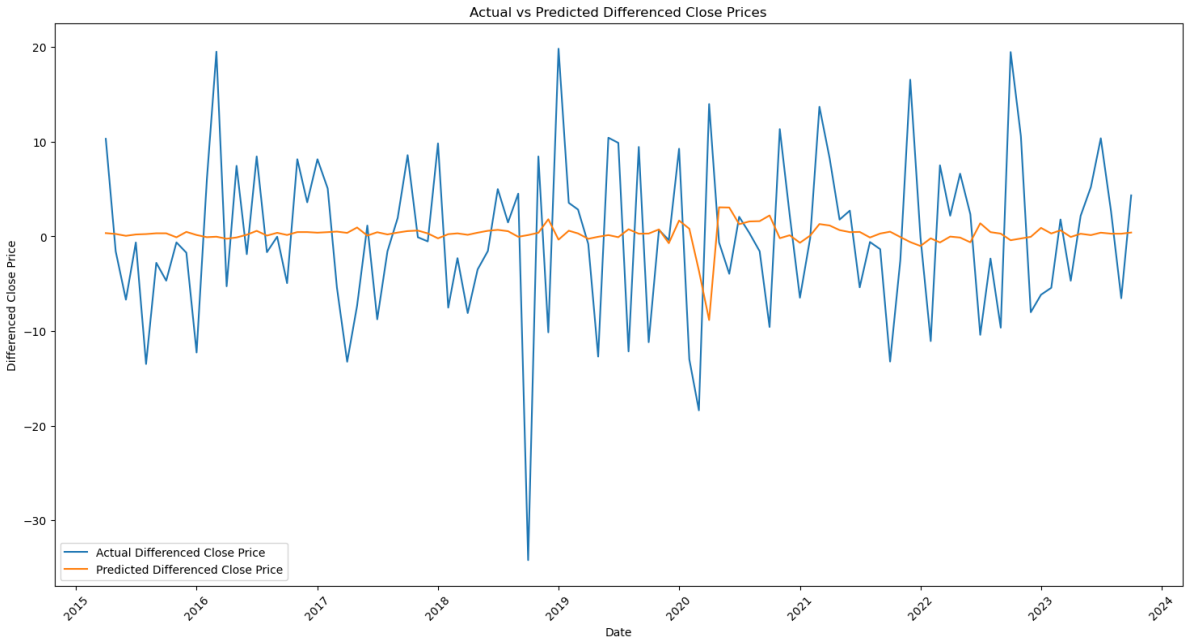
```
print("Mean Absolute Error:",mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)

#Extract the dates from merged_data for the test set for plotting
test_dates = merged_data.index[-len(predictions):]

#Plotting
plt.figure(figsize=(18, 9))
plt.plot(test_dates, y_test_original, label='Actual Differenced Close Price'
plt.plot(test_dates, predictions, label='Predicted Differenced Close Price')
plt.title('Actual vs Predicted Differenced Close Prices')
plt.xlabel('Date')
plt.ylabel('Differenced Close Price')
plt.legend()
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.show()
```

```
Epoch 1/100
19/19 [==============================] - 2s 3ms/step - loss: 0.2810
Epoch 2/100
19/19 [==============================] - 0s 3ms/step - loss: 0.1848
Epoch 3/100
19/19 [==============================] - 0s 3ms/step - loss: 0.1074
Epoch 4/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0530
Epoch 5/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0250
Epoch 6/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0172
Epoch 7/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0165
Epoch 8/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 9/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0165
Epoch 10/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0165
Epoch 11/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 12/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 13/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 14/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 15/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 16/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 17/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 18/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0165
Epoch 19/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 20/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 21/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 22/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 23/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 24/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 25/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 26/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 27/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 28/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 29/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 30/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 31/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 32/100
```

```
19/19 [==============================] - 0s 3ms/step - loss: 0.0167
Epoch 33/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 34/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 35/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0164
Epoch 36/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 37/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0164
Epoch 38/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 39/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 40/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 41/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0162
Epoch 42/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 43/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 44/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 45/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 46/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 47/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0163
Epoch 48/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 49/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 50/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 51/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 52/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 53/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 54/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 55/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 56/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 57/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0162
Epoch 58/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0162
Epoch 59/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0161
Epoch 60/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0162
Epoch 61/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 62/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 63/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
```

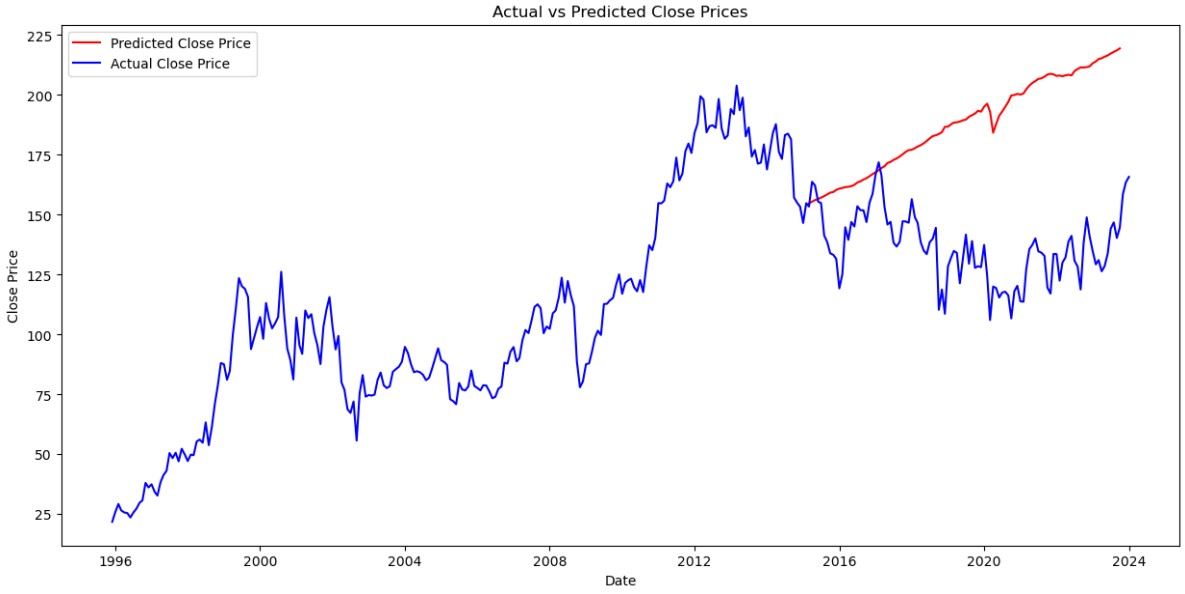
```
Epoch 64/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 65/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 66/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0162
Epoch 67/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 68/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0161
Epoch 69/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0161
Epoch 70/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0161
Epoch 71/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 72/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 73/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 74/100
19/19 [==============================] - 0s 4ms/step - loss: 0.0160
Epoch 75/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 76/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 77/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 78/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 79/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 80/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 81/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 82/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 83/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 84/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 85/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 86/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 87/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 88/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 89/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 90/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 91/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 92/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 93/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 94/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0161
Epoch 95/100
```

```
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 96/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 97/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 98/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 99/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0160
Epoch 100/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
4/4 [==============================] - 1s 3ms/step
Mean Absolute Error: 6.718933966426739
Mean Squared Error: 78.1985011715248
Root Mean Squared Error: 8.842991641493551
```



In [148]:

```
# Start with the last actual close price from the training set
last_train_close = ibm_data.iloc[len(X_train) - 1]['close']

# Reconstruct the predicted 'close' prices
# The predictions are the differenced values, so we need to add these to the
predicted_close_prices = [last_train_close]
for pred_diff in predictions.flatten():
    predicted_close_prices.append(predicted_close_prices[-1] + pred_diff)

# Remove the first element which was the last actual close price
predicted_close_prices.pop(0)

# Extract the test set dates
test_dates = merged_data.index[len(X_train):len(X_train) + len(predicted_clo

# Plotting the actual and predicted close prices
plt.figure(figsize=(15, 7))
plt.plot(test_dates, predicted_close_prices, color='red', label='Predicted C
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Actual Clos
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

```



All four features

In [159]:

```
# Calculate the first difference to make the series stationary
rpietr['RPIETR_diff'] = rpietr['W875RX1'].diff()
rmats['RMATS_diff'] = rmats['CMRMTSPL'].diff()
payems['PAYEMS_diff'] = payems['PAYEMS'].diff()
ipi_index['INDPRO_diff'] = ipi_index['INDPRO'].diff()
ibm_data['close_diff'] = ibm_data['close'].diff()

# Interpolate missing values
rpietr.interpolate(method='linear', inplace=True)
rmats.interpolate(method='linear', inplace=True)
payems.interpolate(method='linear', inplace=True)
ipi_index.interpolate(method='linear', inplace=True)
ibm_data.interpolate(method='linear', inplace=True)

# Drop any remaining NaNs after interpolation
rpietr.dropna(inplace=True)
rmats.dropna(inplace=True)
payems.dropna(inplace=True)
ipi_index.dropna(inplace=True)
ibm_data.dropna(inplace=True)

# Merge the dataframes on their index (DATE)
merged_data = pd.concat([rpietr, rmats, payems, ipi_index, ibm_data], axis=1

# Select features and target
features = merged_data[['RPIETR_diff', 'RMATS_diff', 'PAYEMS_diff', 'INDPRO_
target = merged_data['close_diff'].values

# Initialize the scalers
scaler_x = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))

# Scale the features and target
features_scaled = scaler_x.fit_transform(features)
target_scaled = scaler_y.fit_transform(target.reshape(-1, 1)).flatten()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target_

# Reshape the data for LSTM layer
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=False, input_shape=(X_train.shape[
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=12, verbose=1)

# Predict and inverse transform to original scale
predictions_scaled = model.predict(X_test)
predictions = scaler_y.inverse_transform(predictions_scaled)

#Inverse transform y_test to original scale
y_test_original = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten(

#Calculate MSE and RMSE
mse = mean_squared_error(y_test_original, predictions)
rmse = sqrt(mse)
```

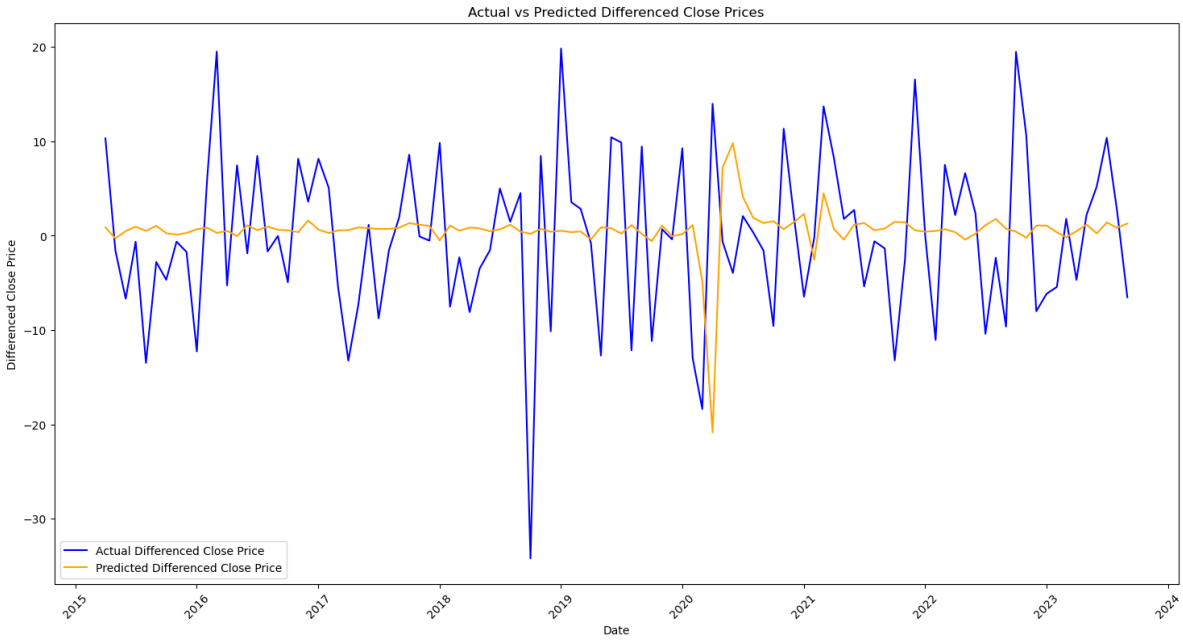
```
mae = mean_absolute_error(y_test_original, predictions)
print("Mean Absolute Error:",mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)

#Plotting
plt.figure(figsize=(18, 9))
test_dates = merged_data.index[-len(predictions):] # Assuming the dates are
plt.plot(test_dates, y_test_original, label='Actual Differenced Close Price'
plt.plot(test_dates, predictions.flatten(), label='Predicted Differenced Clo
plt.title('Actual vs Predicted Differenced Close Prices')
plt.xlabel('Date')
plt.ylabel('Differenced Close Price')
plt.legend()
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.show()
```

```
Epoch 1/100
19/19 [==============================] - 2s 3ms/step - loss: 0.3790
Epoch 2/100
19/19 [==============================] - 0s 3ms/step - loss: 0.1674
Epoch 3/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0534
Epoch 4/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0184
Epoch 5/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0162
Epoch 6/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 7/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 8/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 9/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 10/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 11/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 12/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0159
Epoch 13/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 14/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 15/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 16/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 17/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 18/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 19/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 20/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 21/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 22/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 23/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 24/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 25/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0159
Epoch 26/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 27/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 28/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 29/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0163
Epoch 30/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 31/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 32/100
```

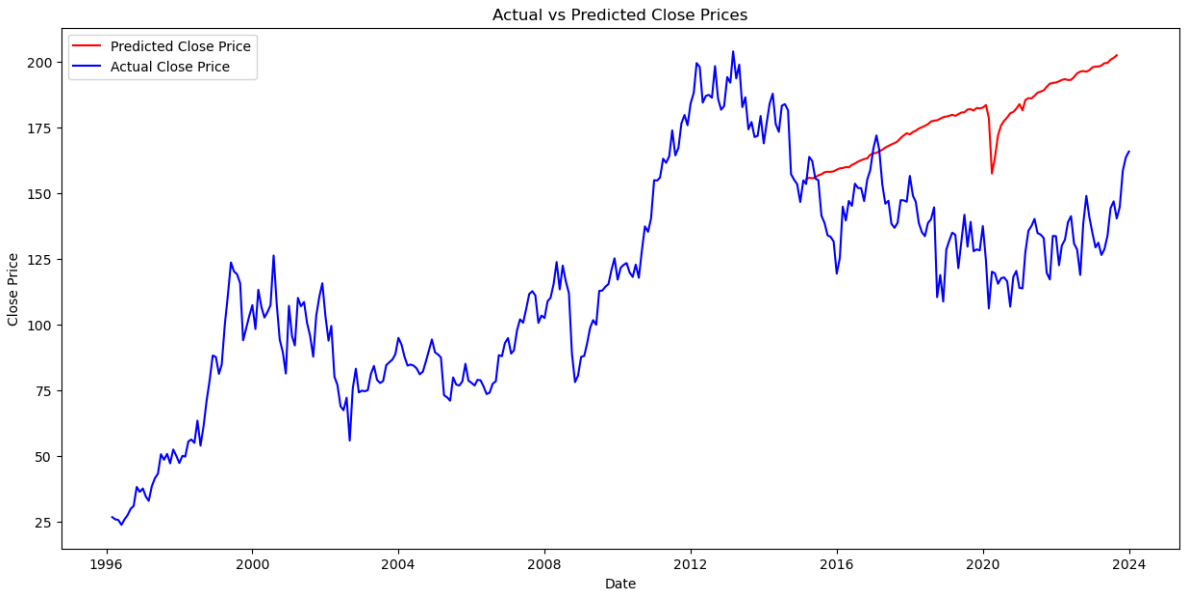
```
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 33/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 34/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 35/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0160
Epoch 36/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 37/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 38/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 39/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 40/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 41/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 42/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 43/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 44/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 45/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 46/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 47/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 48/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 49/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 50/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 51/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 52/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 53/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0159
Epoch 54/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 55/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 56/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 57/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 58/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 59/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 60/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 61/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 62/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 63/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
```

```
Epoch 64/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 65/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 66/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 67/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 68/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 69/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 70/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 71/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 72/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 73/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 74/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 75/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 76/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 77/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 78/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0160
Epoch 79/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 80/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 81/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 82/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 83/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 84/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 85/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 86/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 87/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 88/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 89/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 90/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 91/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 92/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0158
Epoch 93/100
19/19 [==============================] - 0s 2ms/step - loss: 0.0157
Epoch 94/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0158
Epoch 95/100
```

```
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 96/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0156
Epoch 97/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 98/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0159
Epoch 99/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
Epoch 100/100
19/19 [==============================] - 0s 3ms/step - loss: 0.0157
4/4 [==============================] - 1s 2ms/step
Mean Absolute Error: 7.030648796399142
Mean Squared Error: 86.75655650941547
Root Mean Squared Error: 9.314319970315356
```



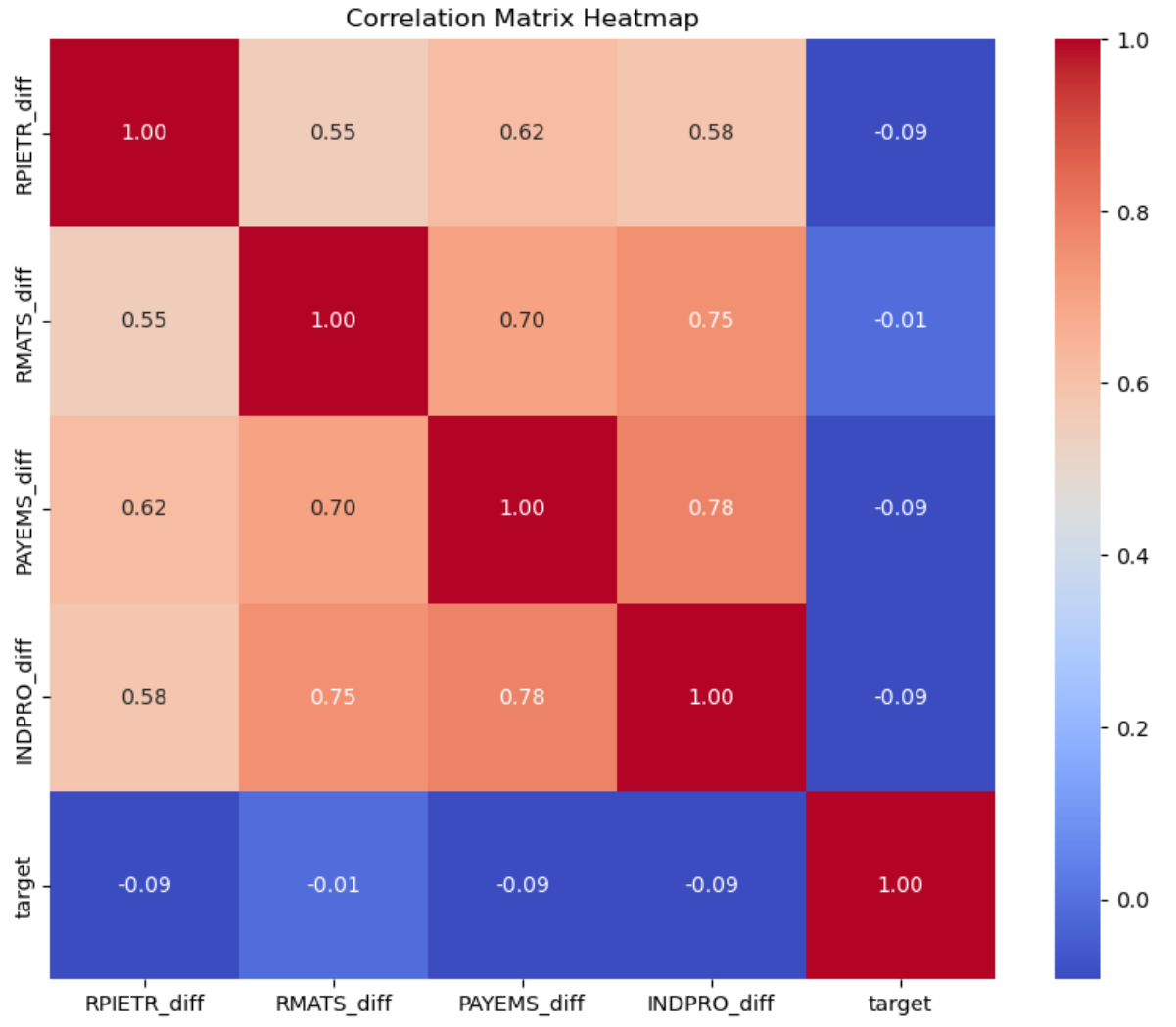
In [154]:
```
# Start with the last actual close price from the training set
last_train_close = ibm_data.iloc[len(X_train) - 1]['close']

# Reconstruct the predicted 'close' prices
# The predictions are the differenced values, so we need to add these to the
predicted_close_prices = [last_train_close]
for pred_diff in predictions.flatten():
    predicted_close_prices.append(predicted_close_prices[-1] + pred_diff)

# Remove the first element which was the last actual close price
predicted_close_prices.pop(0)

# Extract the test set dates
test_dates = merged_data.index[len(X_train):len(X_train) + len(predicted_clo

# Plotting the actual and predicted close prices
plt.figure(figsize=(15, 7))
plt.plot(test_dates, predicted_close_prices, color='red', label='Predicted C
plt.plot(ibm_data.index, ibm_data['close'], color='blue', label='Actual Clos
plt.title('Actual vs Predicted Close Prices')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

```



In [ ]:
```
df_input.to_csv('C:/Users/DeLL/Documents/ibm_data and All three features.csv
```

In [216]:

```
# Assuming rpietr, rmats, payems, ipi_index, and ibm_data are already loaded
# and they all have 'DATE' as their index.

# Merge the dataframes on their index (DATE)
merged_data = pd.concat([rpietr, rmats, payems, ipi_index, ibm_data], axis=1

# Select features and target
features = merged_data[['RPIETR_diff', 'RMATS_diff', 'PAYEMS_diff', 'INDPRO_
target = merged_data['close_diff']

# Create a new DataFrame for the correlation matrix
correlation_data = features.copy()
correlation_data['target'] = target

# Calculate the correlation matrix
corr_matrix = correlation_data.corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()

```



In [ ]:

```
1
```